Synthèse de règles de sécurité pour des systèmes autonomes critiques

Mathilde Machin

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :
l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue le *12/11/2015* **par :**

MATHILDE MACHIN

Synthèse de règles de sécurité pour des systèmes autonomes critiques

---

JURY

CYRIL BRIAND	Professeur des universités	Président du jury
CHARLES PECHEUR	Professeur	Rapporteur
JEAN-MARC THIRIET	Professeur des universités	Rapporteur
ÉRIC RUTTEN	Chargé de recherche	Examinateur
ANTOINE ROLLET	Maître de conférence	Examinateur
HÉLÈNE WAESELYNCK	Directrice de recherche	Directrice de thèse
JÉRÉMIE GUIOCHET	Maître de conférence	Directeur de thèse
JEAN-PAUL BLANQUART	Ingénieur	Co-encadrant de thèse
DAVID POWELL	Directeur de recherche retraité	Invité

École doctorale et spécialité :
EDSYS : Informatique 4200018
Unité de Recherche :
Laboratoire d'analyse et d'architecture des systèmes

À la vie souriante.

Remerciements

Les travaux présentés dans ce mémoire ont été réalisés au *Laboratoire d'Analyse et d'Architecture des Systèmes* du *Centre National de la Recherche Scientifique* (LAAS-CNRS). Je remercie Monsieur Jean Arlat, directeur du LAAS, pour m'avoir reçue au sein de ce laboratoire. Je remercie également Madame Karama Kanoun et Monsieur Mohamed Kaâniche, responsables successifs du groupe *Tolérance aux Fautes et Sûreté de Fonctionnement* (TSF) pour m'avoir accueilli dans ce groupe de recherche. Ces travaux ont été financés grâce à une bourse du Ministère de l'enseignement supérieur et de la recherche, au soutien d'Airbus Defence and Space et au projet européen Saphari-FP7.

Je remercie Monsieur Charles Pecheur, Professeur à l'Université Catholique de Louvain, et Monsieur Jean-Marc Thiriet, Professeur à l'Université Joseph Fourier, qui ont accepté la charge d'être rapporteurs. Je remercie également les autres membres de mon jury de thèse :

— Monsieur Cyril Briand, Professeur à l'Université Paul Sabatier,

— Monsieur Antoine Rollet, maître de conférence à l'ENSEIRB,

— Monsieur Éric Rutten, chargé de recherche à l'INRIA Grenoble.

Je souhaite remercier les personnes avec qui j'ai travaillé durant ces 3 ans, et en premier lieu mes directeurs de thèse, Jérémie Guiochet pour sa confiance et sa compréhension, Hélène Waeselynck pour sa patte formelle, et David Powell avec qui j'ai aimé rédiger un article, à défaut de cette thèse. Je suis reconnaissante à Fanny Dufossé pour notre collaboration fructueuse lors de son séjour post-doctoral. Je remercie Jean-Paul Blanquart, ingénieur chez Airbus Defence and Space, et Matthieu Roy, chargé de recherche, pour leur participation à mes réunions. Lors de la rédaction de cette thèse, j'ai particulièrement apprécié l'aide que m'ont apportée Jean-Paul et Kalou Cabrera-Castillos, post-doctorant, et je les en remercie chaleureusement. Je remercie Steffen Walther et Vito Magnanimo, ingénieurs chez Kuka, pour leur accueil, leur intérêt et leur aide lors de l'étude de cas.

J'ai passé trois ans dans l'équipe TSF, et en particulier parmi les doctorants. Je les remercie d'avoir partagé bonne humeur, adversité et petits gâteaux. Donc, merci à Pierre, Kalou et Yann (dont j'ai aussi apprécié l'aide latexienne), à Camille, à Ivan, à Roberto, à Carla, entre autres. Avec Joris dans l'équipe, je ne peux de toute façon pas viser l'exhaustivité !

De même, j'omets mais n'oublie pas ceux qui m'ont aidé à commencer une thèse, ceux qui m'ont soutenu tout au long de ces trois ans et ceux aux côtés de qui j'ai rédigé, au parc, à la piscine, au petit-déjeuner, sur un court de tennis, dans la jungle de Rome et au ronronnement de la piqueuse.

Sommaire

Introduction

Les progrès scientifiques et technologiques, notamment dans les domaines des capteurs, de l'automatique, de l'intelligence artificielle, permettent de doter les systèmes d'une certaine autonomie. Cette nouvelle capacité a pour but de faire opérer les systèmes dans des environnements plus diversifiés et de fournir de nouveaux services. Cependant, certains de ces services sont sources de questionnements éthiques : guerre sans pertes humaines (pour un camp), assistance déshumanisée aux personnes âgées, responsabilité lors d'un accident impliquant une voiture autonome.

Ce dernier cas illustre que les capacités physiques des systèmes autonomes soulèvent des problèmes de sécurité. Dans de nombreux domaines critiques (avionique, nucléaire, etc), les moyens d'assurer la sécurité ont été étudiés dans le cadre de la sûreté de fonctionnement. Parmi ces moyens, la tolérance aux fautes est adaptée aux environnements dynamiques puisqu'elle permet de tolérer les fautes apparaissant à l'exécution. Une méthode consiste notamment à ajouter au système un moniteur qui le surveille et intervient selon des règles de sécurité.

Dans le cas des systèmes autonomes, élaborer ces règles de sécurité est particulièrement difficile. Les environnements peu structurés rendent les dangers plus complexes et plus nombreux. Par ailleurs, les systèmes autonomes remplissant des tâches diversifiées, ils peuvent causer des dommages diversifiés. Cette multiplication des dangers amène à définir de nombreuses règles de sécurité, qui peuvent être conflictuelles et poser en elles-mêmes des problèmes de sécurité.

En outre, les règles de sécurité peuvent nuire aux fonctions du système. Par l'intermédiaire des interventions du moniteur, les règles pourraient empêcher le système de remplir toute tâche. Il faut donc définir des règles assez fortes pour assurer la sécurité, mais suffisamment permissives pour que le système puisse remplir ses fonctions. La polyvalence attendue des systèmes autonomes exacerbe la difficulté à établir ce compromis.

La diversité des dangers et des services attendus amène à multiplier et à complexifier les règles de sécurité. Cependant, l'augmentation de la complexité d'un mécanisme de sécurité le rend rapidement inutile car la confiance dans sa capacité à assurer la sécurité ne peut plus être justifiée.

Nous proposons donc une méthode systématique pour élaborer et justifier des règles de sécurité qui tiennent compte de la polyvalence des systèmes autonomes. Pour ce faire, nous adoptons une modélisation formelle qui permet d'exprimer les bonnes propriétés attendues des règles. Parmi elles, la sécurité est définie en s'appuyant sur une analyse de risque. Le choix de techniques formelles autorise la vérification automatique des propriétés et la synthèse automatique de règles satisfaisant ces propriétés.

Le premier chapitre de cette thèse présente le contexte de nos travaux. Les systèmes autonomes sont définis et leurs caractéristiques présentées. Des travaux

visant à en assurer la sécurité sont ensuite exposés, suivant la classification des moyens de la sûreté de fonctionnement.

En s'appuyant sur cet état de l'art, le chapitre 2 donne une vue générale de la problématique et de l'approche développée dans ce manuscrit. Cette approche repose sur la formalisation des résultats d'une analyse de risque et permet de synthétiser automatiquement des règles.

Le chapitre 3 détaille le modèle formel sur lequel repose notre méthode. Ce modèle comprend notamment les moyens d'observation et d'intervention du moniteur qui permettent de traiter les dangers. Les règles sont évaluées par l'intermédiaire de propriétés vérifiables automatiquement en utilisant l'outil NuSMV. Le modèle ainsi défini constitue l'entrée de la synthèse. Grâce aux résultats obtenus, le modèle est enrichi par des règles. La vérification de la cohérence des règles synthétisées séparément est présentée.

La synthèse des règles définies sur la base de ce modèle formel fait l'objet du chapitre 4. S'appuyant sur des techniques de vérification formelle, nous proposons un algorithme et un outil de synthèse. Les propriétés des ensembles de solutions retournés ainsi que les performances de l'outil sont analysées. Nous proposons également une approche alternative, reposant sur la théorie des jeux.

Dans le chapitre 5, une étude de cas illustre l'application de la méthode proposée. Le système considéré, fourni par notre partenaire industriel KUKA, est celui d'un robot manufacturier conçu pour collaborer avec des opérateurs humains. Les règles synthétisées ont été implémentées et testées.

En conclusion, nous faisons le bilan de nos contributions avant d'ouvrir sur les perspectives ouvertes par ces travaux.

Chapitre 1

La sécurité-innocuité dans les systèmes autonomes

Sommaire

> «La sûreté de fonctionnement d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre.»
>
> [Laprie 1996]

La sécurité-innocuité est l'attribut de la sûreté de fonctionnement défini comme «la non-occurrence de conséquences catastrophiques pour l'environnement» dues au système [Laprie 1996] [Avižienis 2004]. Les conséquences catastrophiques comprennent, bien évidemment, le décès et les blessures des utilisateurs et des opérateurs, mais aussi des dommages matériels et financiers, s'ils sont sans commune

mesure avec le service fourni par le système. Beaucoup de systèmes ne sont pas concernés par cette propriété car leurs limites physiques ne leur permettent pas de causer de tels dommages. Les autres systèmes sont appelés *systèmes critiques*. On abrégera dans la suite *sécurité-innocuité* par *sécurité*.

La sécurité entretient des relations paradoxales avec la continuité du service correct, caractérisée en sûreté de fonctionnement par la fiabilité et la disponibilité. D'une part, la continuité de service participe à la sécurité : des défaillances récurrentes suscitent des usages incorrects, potentiellement dangereux. D'autre part, sécurité et continuité de service peuvent être antagonistes. Un système de sécurité est traditionnellement responsable de l'arrêt d'urgence, c'est-à-dire de mettre le système dans un état stable et non-fonctionnel, supposé être sûr. L'arrêt empêche ou diminue les dommages mais ne permet pas de fournir le service attendu. Toutefois l'arrêt n'est pas nécessairement l'état le plus sûr. Par exemple, le freinage d'urgence d'une voiture soulève des problèmes de sécurité. L'arrêt est très utilisé car c'est un état qui peut être atteint avec une grande fiabilité. C'est la troisième interaction entre sécurité et continuité de service : la sécurité ne peut être assurée que par un système de sécurité fiable, capable de fournir un service correct même dans des situations exceptionnelles.

Parmi les systèmes critiques, les systèmes critiques autonomes forment une classe apparue récemment. Après les avoir présentés dans la section 1.1, on présentera comment les différents moyens de la sûreté de fonctionnement sont employés pour assurer la sécurité des systèmes autonomes.

La prévention des fautes ou comment empêcher l'occurrence ou l'introduction des fautes. La prévention se traduit par exemple par des méthodes de développement rigoureuses, mais elle ne se distingue pas de la démarche d'ingénierie. Elle ne sera donc pas traitée en tant que telle mais seulement abordée dans la section 1.1.2.

La prévision des fautes ou comment estimer la présence et les possibles conséquences des fautes. Pour les systèmes autonomes, les travaux se concentrent sur les techniques d'analyse de risque, qui constituent la section 1.2.

L'élimination des fautes ou comment réduire la présence (nombre, sévérité) des fautes, introduites malgré l'utilisation de la prévention. Les techniques d'élimination sont développées dans la section 1.3.

La tolérance aux fautes ou comment fournir un service à même de remplir la fonction du système en dépit des fautes présentes dans le système. Ces fautes sont soit des fautes résiduelles, qui subsistent après l'utilisation des techniques précédentes, soit des fautes apparaissant pendant la vie opérationnelle, telles que les fautes d'interaction. Des exemples de tolérance aux fautes sont exposés à la section 1.4.

1.1 Les systèmes autonomes critiques

Cette section s'attache à présenter les caractéristiques des systèmes autonomes et à identifier leurs spécificités vis-à-vis de la sécurité. Notre étude porte sur les systèmes autonomes critiques.

1.1.1 Définition et caractéristiques des systèmes autonomes

Il n'existe pas de consensus autour d'une définition de l'autonomie des systèmes. Certaines se rapprochent du sens étymologique : «se gouverner d'après ses propres lois». Une définition plus technique et qui reflète plus la réalité de la robotique autonome a été proposée par le Nist (*National Institute of Standards and Technology*) [Huang 2008] :

> «L'autonomie est la capacité d'un système à percevoir, analyser, communiquer, planifier, établir des décisions et agir, afin d'atteindre des objectifs assignés par un opérateur humain.»

Cette définition n'établit pas de distinction franche entre les systèmes autonomes et les systèmes automatiques. En effet, l'autonomie est une propriété graduelle que le Nist propose d'évaluer par la complexité et la variété des missions, la difficulté de l'environnement et les interactions homme-robot. Des éléments sont communs à tous les systèmes physiques : la perception, l'actionnement. La commande, décomposée en analyse, communication, planification et décision est largement plus développée que dans les systèmes automatiques. La capacité d'analyse suppose différents niveaux d'abstraction et la planification une vision à long terme. Par exemple, la commande d'une voiture autonome doit mettre en œuvre le contrôle moteur mais aussi tenir compte des piétons et du code la route, tout en planifiant le trajet sur plusieurs kilomètres. Un déambulateur robotisé qui aide le patient à s'équilibrer pendant la marche a un degré d'autonomie moindre car il ne peut que rejoindre sa base de manière autonome. D'autre part, son environnement, l'hôpital, est bien moins accidenté.

La complexité de la commande permet au système de remplir des missions plus complexes et/ou plus diversifiées, sans intervention de l'opérateur, dans un environnement ouvert, peu ou pas structuré. S'adapter à l'environnement nécessite une certaine puissance physique et une latitude de mouvement, ce qui donne aussi au système les moyens de causer des dommages. C'est pourquoi les systèmes autonomes sont, pour la plupart, critiques.

Les systèmes autonomes ne sont pas tous robotiques, citons par exemple les voitures électriques. Par ailleurs la robotique est loin de se limiter aux systèmes autonomes. Toutefois une grande part des systèmes autonomes relève de la robotique, et les techniques propres à l'autonomie, telles que la planification, sont issues de la communauté de la robotique. Même si la définition de l'autonomie ne fait pas référence à la robotique, l'autonomie y est historiquement et technologiquement liée.

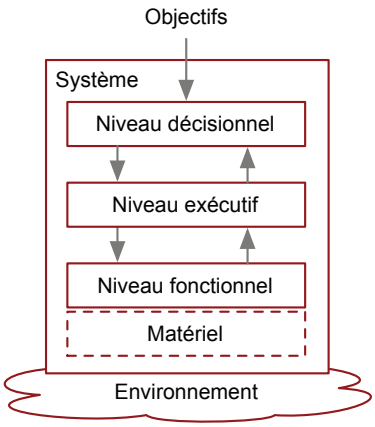


FIGURE 1.1 – Architecture hiérarchisée

La robotique automatique a répondu au problème de la sécurité par des mesures structurant l'environnement. D'une part, l'environnement est figé ou normalisé pour être connu et exempt de situations adverses. Un robot manufacturier dont la trajectoire est complètement programmée à la conception ne peut saisir une pièce que si elle arrive précisément à l'endroit prévu. Dans le cas de la navigation dans une usine, des rails peuvent être mis en place, ou une carte des obstacles (qui sont donc considérés comme fixes) établie ; ou encore les obstacles peuvent être modifiés afin qu'ils soient détectables à la hauteur des capteurs laser des robots. D'autre part, l'espace est réparti entre les robots et les humains ; la délimitation a été marquée par des barrières physiques puis des rideaux infra-rouges. Dans le cas des systèmes autonomes, il est au contraire souhaitable que le système puisse évoluer dans un environnement le moins structuré possible, et en particulier en milieu humain. Dans ce contexte, l'enjeu est évidemment de déterminer les mesures de sécurité permettant cette évolution.

1.1.2 Architecture et outils de développement

Plusieurs types d'architectures ont été proposés pour la commande des systèmes autonomes. Une présentation en est par exemple faite dans [Lussier 2007a]. Cependant, une grande majorité des systèmes autonomes ont maintenant une architecture de type hiérarchisé, et nous nous y limiterons. Comme illustré dans la figure 1.1, l'architecture hiérarchisée est classiquement structurée en trois niveaux :

Le niveau décisionnel est le plus haut niveau de l'architecture. Il reçoit des objectifs et génère des plans à partir d'une représentation abstraite du système et de son environnement. Ce niveau n'offre pas de garantie en terme de temps d'exécution.

Le niveau exécutif convertit les plans qu'il reçoit du niveau décisionnel en fonctions élémentaires du niveau fonctionnel. Ce niveau a des exigences temps-réel.

Le niveau fonctionnel gère l'interface matérielle. Le niveau est décomposé en fonc-

tions élémentaires qui communiquent entre elles mais ne partagent pas de représentation globale du système. Le temps d'exécution doit être adapté à l'interface avec les capteurs et les actionneurs.

De plus, chaque niveau rend compte au niveau supérieur de l'accomplissement des tâches et des éventuelles erreurs qui ne pourraient pas être traitées localement. L'architecture hiérarchisée existe en de nombreuses versions, par exemple sans niveau décisionnel ou avec des niveaux exécutif et fonctionnel fusionnés. Les architectures hybrides permettent aux observations du niveau fonctionnel d'être propagées directement au niveau décisionnel, pour accélérer leur prise en compte.

L'hétérogénéité au sein de l'architecture soulève des problèmes de communication entre les différents niveaux et fonctions. C'est pourquoi la communication entre composants logiciels est réalisée par des intergiciels (ou *middleware*), tels que ROS (*Robot Operating System*, [Quigley 2009]). Les intergiciels facilitent aussi la réutilisation et l'échange des composants développés, et contribuent à la qualité du logiciel en fournissant des services et des formats standardisés.

Dans un objectif d'efficacité et de prévention des fautes, des environnements de développement propres à la robotique existent. Par exemple, GenoM ([Fleury 1997], [Mallet 2010]) permet d'abstraire le niveau fonctionnel et de rendre modulaire chaque fonction. Pour chaque composant, un minimum de code est écrit par le développeur, la gestion de la communication entre les composants étant générée automatiquement.

1.1.3 Interactions avec l'homme

Une caractéristique importante des systèmes autonomes est liée aux interactions avec l'homme. À l'opposé du robot solitaire sur Mars, un défi majeur de la robotique autonome actuelle est de faire cohabiter et collaborer des robots avec des êtres humains.

La sécurité est très représentée dans les rôles que peut remplir un homme vis-à-vis d'un système autonome en opération [Alexander 2009a] : l'être humain peut être responsable de la sécurité, en appliquant des procédures normalisées mais aussi en faisant face aux situations inattendues, voire en secourant le robot. Les interactions de ce type sont un moyen de traitement des dangers et sont communes à tous les systèmes complexes.

Les interactions physiques[Alami 2006], plus spécifiques aux systèmes autonomes, sont sources de dangers. Elles sont définies comme des contacts entre un être humain et une partie physique du système. Pour réduire la gravité de ces interactions, des robots à faible raideur et capables de percevoir les forces externes ont été développés (voir par exemple [Tonietti 2005]). Les interactions physiques deviennent alors propices à la communication entre l'être humain et le système. Pour assurer la sécurité, la démarche classique de séparation spatiale homme-robot n'est alors plus valide, puisque le contact est désiré.

Au contraire des interactions physiques, les interactions cognitives se font sans contact ; elles passent en général par la vision ou le son. Par exemple, un robot

transportant un couteau pointant vers l'avant est considéré comme dangereux par un utilisateur. Pour y remédier, des approches de planification prenant en compte la présence humaine ont été développées, dont [Sisbot 2012]. Ainsi, la sécurité est prise en compte comme une propriété ressentie par l'utilisateur. Cet aspect s'insère dans le contexte plus large de la confiance [Schaeffer 2013] et de l'acceptabilité d'un système autonome. Dans [Kruse 2013], l'acceptabilité est déclinée en trois paramètres :

— Le confort est l'absence de perturbation ou de stress provoqués par le robot. Cela comprend des limites de distances, de vitesse, d'accélération modulées, par la position de l'utilisateur (assis ou debout, de face ou de dos, etc.).

— Le naturel est la ressemblance du robot avec l'humain. Des mouvements naturels sont notamment plus prévisibles.

— La sociabilité est le respect des conventions culturelles. Par exemple, on évite de passer à travers un groupe.

Le sentiment de sécurité des utilisateurs est assez différent et décorrélé de la sécurité telle qu'elle est envisagée par les experts et les normes.

1.1.4 Normes de sécurité en vigueur

Les systèmes autonomes n'ont pas, à ce jour, de norme qui leur soit propre. Ils sont soumis aux normes existantes en fonction de leur domaine d'application. Ainsi dans le domaine médical, un système autonome sera un «dispositif médical actif», au même titre que tout dispositif électronique en interaction physique avec les patients. Dans le domaine industriel , il sera un «robot industriel» [ISO10218-1 2006], comme l'est un système purement automatique. Ces normes ne prennent pas en compte la dimension décisionnelle de ces systèmes.

Mais le contexte normatif est en train de s'adapter. La norme [ISO13482 2014] pour les robots d'assistance personnelle est sortie tout récemment. La commission ISO/IEC TC184/SC5/JWG9 travaille à une norme pour la robotique médicale. Le guide technique TS15066 pour la collaboration homme-robot industriel est en cours de rédaction. Ce guide contiendra notamment des valeurs maximales de forces et de couples lors des contacts physiques homme-robot, qui sont issues de recherches récentes. Le DLR (*Deutsches Zentrum für Luft- und Raumfahrt*, l'agence spatiale allemande), entre autres, a mis en place une plate-forme d'expérimentation pour déterminer les dommages causés sur un tissu animal par une force donnée (cf. [Haddadin 2011] [Haddadin 2012]).

Les normes pour la robotique sont donc en pleine mutation, tant sur la définition des périmètres des normes, que sur la caractérisation des dangers.

1.1.5 Fautes des systèmes autonomes

Les fautes sont les sources des défaillances. Une fois activée, une *faute* produit une *erreur*, qui, si elle n'est pas traitée peut devenir perceptible pour l'utilisateur, en

amenant le système à violer sa spécification et à ne pas fournir le service attendu, ce qui forme une *défaillance* (catastrophique ou non). Les fautes sont très diverses : elles peuvent notamment être internes ou externes, introduites lors de la conception ou en opération, être dues à un phénomène physique ou une action humaine. Nous excluons du champ de l'étude les fautes malveillantes.

Les systèmes autonomes évoluant dans des environnements peu structurés, ils sont particulièrement soumis aux fautes externes, aussi appelées *situations adverses*. Les interactions homme-robot donnent également lieu à d'autant plus de fautes que les utilisateurs sont peu formés, peu attentifs, comme ce serait le cas dans un espace grand public.

L'autonomie se caractérisant aussi par une commande complexe, les fautes de conception logicielle sont très représentées. La RoboCup, une compétition entre laboratoires de recherche, a fait l'objet d'un recensement des fautes internes survenues sur 17 robots autonomes [Steinbauer 2013]. Les fautes logicielles sont plus fréquentes que les fautes matérielles et interviennent autant dans les systèmes d'exploitation et intergiciels que dans les algorithmes applicatifs (comme la localisation, la planification, etc). En ce qui concerne les aspects matériels, les capteurs sont autant sujets aux fautes que les actionneurs mais les défaillances sont plus graves, ce qui révèle l'importance de l'observation de l'environnement.

Ce type d'étude, encore rare pour les systèmes autonomes, synthétise les expériences et permet d'enrichir l'analyse de risque, notamment en illustrant les conséquences que peut avoir une défaillance et donc sa gravité. Les retours d'expérience peuvent ainsi révéler des problèmes qui seront ensuite systématiquement examinés pour les systèmes. La capitalisation des retours d'expériences se fait notamment par des listes de dangers. Cette approche, très statique, est efficace mais insuffisante pour les systèmes autonomes, qui évoluent dans des environnements changeants. Elle peut être utilisée en complément d'une analyse de risque plus ouverte.

1.2 Prévision des fautes

La prévision des fautes englobe des techniques d'analyse de risque et des techniques d'évaluation probabiliste, purement quantitative . L'évaluation quantitative n'est pour l'instant pas appliquée aux systèmes autonomes en raison de leur complexité et du manque de recul technologique. L'analyse de risque est définie par le [Guide ISO/IEC 51 1999] comme «*l'utilisation systématique des informations disponibles pour identifier les dangers et estimer le risque*». Le risque est estimé en fonction de la probabilité d'occurrence et de la gravité des dommages. Ces deux paramètres sont généralement évalués par des niveaux discrets tels que «*Faible*», «*Elevé*». Les analyses de risque sont conduites à différents niveaux d'abstraction et à différentes étapes du développement.

Les techniques d'analyses de risque se décomposent en deux familles : les techniques, comme les arbres de fautes, qui d'un évènement redouté, remontent aux

fautes pouvant le causer ; et les techniques qui suivent le sens causal, en déterminant les conséquences de chaque faute potentielle telles que l'Amdec (*Analyse des Modes de Défaillances, de leurs Effets et de leur Criticité*, l'Hazop (*Hazard Operability analysis*, étude de danger et d'opérabilité). Les techniques inductives, qui supposent de connaître les fautes à considérer, sont parfois difficiles à mettre en œuvre. La spécificité de l'Hazop est d'accompagner l'identification des fautes à considérer : des mots-guides génériques (par exemple «trop de», «absence de») sont appliqués à des entités du système. Dans une table Hazop, comme par exemple le tableau 1.1, l'association d'une entité et d'un mot-guide permet d'imaginer une déviation dans le système. Ses causes et conséquences, notamment en terme de gravité, sont ensuite évaluées et des mesures pour l'éviter ou le maîtriser sont proposées selon les besoins et les possibilités. L'évaluation de la probabilité d'occurrence peut être ajoutée.

Les méthodes d'analyse de risque sont informelles et ne peuvent prétendre à une couverture totale des risques. Cependant, elles sont systématiques et correspondent donc à la plus grande couverture possible. En raison des risques atypiques que l'autonomie suppose, des méthodes d'analyses spécifiques ont été développées.

Traditionnellement, l'analyse porte sur le système décomposé en sous-systèmes. Dans [Böhm 2010], l'analyse de risque d'un robot-jouet thérapeutique s'appuie sur deux décompositions différentes du système : les composants et les fonctionnalités. La technique choisie est l'Hazop car elle permet d'avoir une bonne couverture, y compris pour des systèmes novateurs. Cependant, l'environnement et les interactions, aspects caractéristiques des systèmes autonomes, ne sont pas spécifiquement analysés.

Dans [Alexander 2009b], les auteurs proposent d'associer plusieurs méthodes existantes. Des listes de dangers issues de l'expérience sont utilisées ainsi que la méthode ETBA (*Energy Trace and Barrier Analysis*). L'ETBA part du principe qu'une défaillance catastrophique est une libération d'énergie non-désirée. Ainsi les dangers sont d'abord considérés comme des évènements physiques, avant d'être envisagés comme des résultats d'une chaine de causalité ayant pour origine des fautes . Puis deux méthodes d'analyse de risque (Hazop et FFA *Functional Failure Analysis*) sont utilisées pour analyser les fonctions et les flux de données au sein du système. Enfin, les résultats, et notamment les liens de causalité entre les défaillances, sont organisés dans un arbre de fautes, ce qui permet le calcul des coupes minimales. L'approche consistant à combiner des techniques pour obtenir une couverture maximale est intéressante mais la procédure proposée est lourde.

Plus simplement, dans [Woodman 2012], une variante de l'Hazop spécialisée pour le logiciel, Hazop-Shard (*Software Hazard Analysis and Resolution in Design*) est associée avec des listes de dangers pour l'environnement. L'expérience manque pour arguer de la couverture de ces listes pour une classe de systèmes aussi récente. En outre, les listes ne sont pas suffisantes pour analyser les environnements ouverts et non-structurés auquels un système autonome doit faire face.

Prenant note de l'importance de l'environnement dans l'autonomie, les auteurs

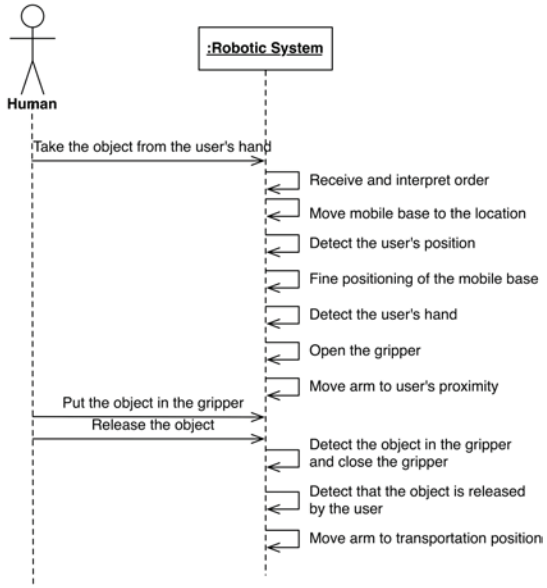


FIGURE 1.2 – Diagramme de séquence UML d'un robot manipulateur mobile [Martin-Guillerez 2010]

de [Dogramadzi 2014] ont développé une méthode spécifique pour analyser l'environnement seul, à associer avec une méthode classique pour analyser le système. L'idée est d'analyser l'environnement en dehors du cadre des tâches du système. En effet les interactions inattendues sont à la fois dangereuses et difficiles à mettre au jour par des méthodes classiques d'analyse. L'environnement, êtres vivants inclus, est décomposé en agents, objets et éléments d'arrière-plan, eux-mêmes catégorisés, ce qui permet une décomposition fine et, autant que faire se peut, sans omission. Les éléments de l'environnement sont ensuite analysés par des tables s'inspirant de l'HAZOP. Les mots-clés sont du type «interaction partielle», «interaction inattendue». Les déviations et leurs conséquences sont déterminées, mais pas la gravité.

Enfin, dans [Martin-Guillerez 2010], les cas d'utilisation du système sont modélisés en UML (*Unified Modeling Language*). Les diagrammes de cas d'utilisation et de séquences sont ensuite analysés par HAZOP. La figure 1.2 montre un diagramme de séquence, dont l'analyse HAZOP est illustrée par le tableau 1.1. L'UML est un bon outil de communication entre les experts du système et les experts en sécurité. Les diagrammes de séquences permettent de considérer les interactions avec l'environnement dans le contexte d'un scénario d'exécution. Le scénario peut paraître restrictif et moins couvrant qu'une analyse générale. Néanmoins, avoir un contexte précis permet d'imaginer des dangers à la fois dans le contexte et hors du contexte, ce qui enrichit l'analyse. L'application de la technique à un robot mobile manipulateur et à un déambulateur robotisé démontre la couverture de cette méthode.

Project : PHRIENDS — Date: June-01-2008
HAZOP number : UC4/SD4 — Prepared by: Ofaina Taofifenua
Entity : Sequence Diagram 4 (sd4) "Take an object from the user's hand" — Revised by: Jérémie Guiochet
Approved by:

Element (attribute)	Guide word	Deviation	a. Use Case Effect b. Real World Effect	Severity	Possible Causes	Integrity level Requirements	New Safety Requirements	Remarks	Hazard Number
Receive and interpret order (pred/succ)	More than / as well as	The robot receives several different orders	a. Wrong order taken into account b. Wrong task, bad synchro-nization between robot and user, could result in collision	**Moderate**	Failure of H/W for order reception Human error	H/W for order reception should be SIL1	User education and training Define a protocol for communication between user and robot (e.g. acknowledgment messages, user can check interpretation of the order)	Means for communication between robot and user needs to be defined for the PHRIENDS use case (speech, graphical HMI, vision, etc.)	
Put the object in the gripper (pred/succ)	Before	Since the gripper is open the user can give the object to the robot before the latter is ready	a. Bad synchronization between user and robot can cause collision b. The object can fall / The arm and human can collide	**Severe**	Human error	None	The robot should keep the gripper closed until the arm movement is finished	The procedure in the seq. diag. is as follows: the robot opens its gripper then the robot arm moves towards the user hand. Only then the user can place the object in the robot gripper. A safer procedure is: the robot should keep the gripper closed until arm movement is finished -> modify sequence diagram	2, 19, 20

Tableau 1.1 – Extrait de la table Hazop analysant le diagramme de séquence [Martin-Guillerez 2010]

1.3 Elimination des fautes

L'élimination des fautes vise à révéler puis diagnostiquer et corriger les fautes introduites dans le système considéré. Seule la première étape est traitée ici, les autres sont beaucoup plus spécifiques au système considéré et moins traitées dans des travaux génériques. Les méthodes d'élimination sont classées en deux catégories : le test, ou vérification dynamique, qui nécessite d'exécuter le système et la vérification statique, qui s'appuie sur un modèle du système. Cette séparation ne doit pas cacher que les techniques employées sont souvent transverses et réutilisables au sein et au-delà de l'élimination des fautes. Par exemple, la génération automatique de cas de tests s'appuie souvent sur des techniques de vérification de modèles.

1.3.1 Test

Le test peut se faire sur toutes les abstractions (modèles, logiciels ou matériels) et sur toutes les décompositions (systèmes, sous-systèmes, composants). Le test est le moyen le plus intuitif pour révéler une faute : on soumet un cas de test aux entrées du système ; ses sorties sont analysées pour déterminer si elles sont correctes. Il faut donc d'abord élaborer l'ensemble des cas de test, dont la qualité est évaluée par des critères de couverture. Les résultats du test sont ensuite analysés, ce qui constitue le problème de l'oracle. Lorsqu'on dispose d'un modèle de comportement complet, il sert d'oracle : la sortie de l'oracle et du système sous test sont comparées. Sinon, on utilise un oracle partiel, qui ne vérifie que certaines propriétés des sorties. Le test peut poursuivre deux objectifs différents. Le test de conformité vise à révéler des fautes. Pour chaque cas de test, l'oracle détermine si les sorties sont conformes ou non. Le test de robustesse vise à évaluer la résistance du système à des conditions environnementales stressantes.

Dans [Micskei 2012], un cadre pour tester la robustesse de la commande logicielle d'un système autonome est présenté. La commande est considérée comme une boite noire. Les tâches du système sont modélisées par des diagrammes de séquences UML et l'environnement est représenté par un méta-modèle (issu d'une ontologie). Cette vue permet d'abstraire l'environnement et donc de prendre en compte sa variabilité. Les modèles permettent de générer les cas de test. Un oracle qui déterminerait les sorties exactes est ici infaisable : d'abord parce qu'il serait aussi complexe que le système sous test, donc contenant potentiellement autant de fautes, ensuite parce que le niveau décisionnel d'un système autonome est typiquement non-déterministe. L'oracle est donc un oracle partiel : les propriétés à vérifier sont constituées par deux diagrammes de séquence. Lors de l'exécution d'un cas de test (sur un système réel ou en simulation), si les messages du premier diagramme sont observés, les messages du second diagramme doivent être observés [Horányi 2013].

Un algorithme d'évitement pour des drones est testé en simulation dans [Zou 2014]. La génération de cas dangereux est faite par une optimisation génétique. L'oracle est ici la distance minimale des deux drones, qui, en dessous d'un certain seuil, est considérée comme une collision. Comme l'objectif de test est la non-collision, une fonction d'adaptation est facilement déterminée et la recherche par algorithme génétique permet d'avoir une bonne couverture. Pour d'autres objectifs de test, et notamment des objectifs plus complexes, la détermination de la fonction d'adaptation serait une difficulté.

Le test de robustesse de la couche fonctionnelle est l'objet des travaux de [Chu 2011] et de [Powell 2012], par exemple. Les cas de test sont obtenus par injection aléatoire de fautes temporelles. Typiquement les requêtes du niveau décisionnel sont retardés. L'oracle est, là aussi, constitué de propriétés, comme par exemple l'exclusion mutuelle de l'activité de deux modules fonctionnels. Les tests sont exécutés dans un environnement simulé.

La simulation est très répandue en robotique. Dans le cadre du test, elle permet d'automatiser les expériences, de les multiplier sans problèmes de coût, d'équipements, ou de sécurité. Cependant un test par simulation n'est valide que dans la mesure où le modèle simulé l'est aussi.

Les systèmes autonomes peuvent également être testés au moyen de la vérification à l'exécution (*runtime verification* en anglais, voir [Leucker 2009] pour un aperçu, et [Delgado 2004] pour une vue plus complète). Cette technique génère un oracle à partir des propriétés, souvent temporelles, spécifiées par exemple par annotation du code. L'oracle, dans son exécution, n'est souvent pas indépendant, ou isolé, du système sous test. Si la vérification se fait pendant l'exécution, et non à partir de traces d'exécution, l'oracle peut aussi être utilisé pendant la vie opérationnelle du système et donc relever de la tolérance aux fautes. Issue de la communauté des méthodes formelles et historiquement tournée vers le logiciel, la vérification à l'exécution a été appliquée aux systèmes physiques (cf. [Goodloe 2010] et, par exemple, [Kane 2014]). Dans [Goldberg 2005], elle est utilisée pour le test de non-régression de la planification d'un système autonome.

1.3.2 Vérification statique

Contrairement au test, les techniques de vérification statique garantissent que toutes les exécutions d'un système sont correctes vis-à-vis des propriétés spécifiées, et reposent obligatoirement sur un modèle du système (fût-ce du code). Les techniques de vérification statique sont la preuve de théorème, la vérification de modèle et l'analyse statique. Cette dernière vérifie des propriétés génériques, comme les débordements de la pile. Comme nous sommes plus intéressés par des propriétés spécifiques aux applications, nous ne développerons pas plus avant cette technique.

La preuve de théorème utilise un système d'inférence logique pour construire une démonstration prouvant que le système satisfait une propriété. Elle peut être utilisée pour prouver des propriétés d'automatique, comme par exemple la stabilité dans [Araiza-Illan 2014]. Un algorithme de navigation, l'évitement d'obstacle fixe, est prouvé par [Täubig 2012]. La preuve est assez peu utilisée, car elle est loin d'être automatique et nécessite des compétences spécifiques.

La vérification de modèle vérifie exhaustivement que les traces (équivalents des exécutions) satisfont des propriétés, généralement définies en logique temporelle. Quelques approches de vérification de modèle établissent le niveau de probabilité avec lequel la propriété est satisfaite (voir par exemple [Pathak 2013], [O'Brien 2014]), elles s'intéressent en majorité à la satisfaction simple de la propriété. Le logiciel de commande bas niveau d'un robot mobile est vérifié par [Scherer 2005]. La vérification porte directement sur le code Java, ce qui évite l'étape de modélisation. L'environnement, à savoir les actionneurs, doivent, eux, être modélisés. Dans [Simmons 2000], les auteurs présentent la vérification de la décomposition et de la synchronisation de tâches en C++ par l'utilisation du model-checker NuSMV. Plus largement, la planification et la vérification ont largement contribué l'une à l'autre, comme exposé dans [Bensalem 2014]. Intuitivement, une fois que l'on est capable de vérifier automatiquement un plan, un contrôleur ou toute autre entité, vis-à-vis d'une propriété, la synthétiser devient théoriquement possible, ne serait-ce que par énumération des entités possibles et vérification des propriétés. On passe ainsi de l'élimination des fautes à une forme avancée de prévention des fautes.

Au vu de la complexité des systèmes, l'élimination laisse des fautes résiduelles à tolérer. De plus, les fautes d'environnement et d'interaction n'existent que durant la vie opérationnelle, elles ne peuvent donc pas être éliminées à la conception, ce qui oblige à les tolérer.

1.4 Tolérance aux fautes

La tolérance aux fautes est utilisée pendant la vie opérationnelle du système pour maintenir un service correct malgré la présence de fautes, et en particulier lors de l'occurrence d'erreurs. On n'abordera pas ici les mesures de tolérance passive qui concernent surtout la conception mécanique : une forme qui empêche le

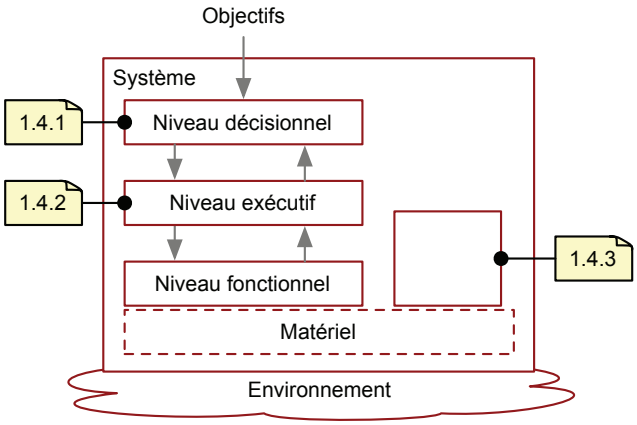


FIGURE 1.3 – Les mécanismes de tolérance situés dans l'architecture hiérarchisée

coincement des doigts, un revêtement souple, etc. Les deux étapes majeures de la tolérance aux fautes active sont la détection d'une erreur et le recouvrement (auxquelles peuvent être ajoutés le diagnostic et l'isolation). La détection implique que le mécanisme mettant en œuvre la tolérance aux fautes a des moyens d'observation, et le recouvrement, qu'il a des moyens d'intervention sur le système. Comme pour l'élimination des fautes, la vérification exacte des sorties n'est pas possible, on utilise donc des propriétés pour détecter les fautes.

L'évaluation ou la vérification de la tolérance aux fautes se fait par l'injection de fautes. Les fautes résiduelles ne se manifestant que très rarement, il est nécessaire d'injecter des fautes pour tester la tolérance aux fautes.

Pour réaliser la tolérance aux fautes, le principe de base consiste à utiliser la redondance. Des capteurs redondants permettent, par exemple, de détecter des incohérences de valeur et donc des défaillances d'un capteur. Comme le résume la figure 1.3, les mécanismes de tolérance aux fautes peuvent être implémentés en différents points d'une architecture hiérarchisée de système autonome. Les mécanismes du niveau décisionnel sont exposés par la section 1.4.1, et ceux du niveau exécutif par la section 1.4.2. Nous ne présenterons pas les mécanismes de tolérance aux fautes implémentés dans le niveau fonctionnel car ils sont souvent très simples et liés à la couche matérielle. Un exemple classique pour un robot mobile est le pare-choc qui une fois enfoncé coupe l'alimentation de moteurs. Une autre approche consiste à séparer l'architecture de commande du mécanisme de sécurité, comme exposé au section 1.4.3. Enfin, l'identification des spécifications du mécanisme de sécurité est discutée dans la section 1.4.4

1.4.1 Mécanismes dans le niveau décisionnel

Dans [Ertle 2010], le niveau décisionnel couvre les fautes d'environnement. Le planificateur a une représentation des états atteignables du système. Si les états violent des propriétés de sécurité données, ils sont considérés comme dangereux. Le risque, représenté par un nombre entre 0 et 1, est propagé graduellement aux

états intermédiaires entre les états dangereux et l'état courant. Ainsi le planificateur prend en compte la distance au danger (une distance pas seulement spatiale mais généralisée à toute grandeur définissant un danger, par exemple la température). L'identification des dangers se fait hors ligne, et l'évaluation du risque en ligne.

Du point de vue de la commande logicielle, les fautes d'environnement sont très semblables aux fautes des capteurs et des actionneurs, et sont souvent traitées ensemble. Dans [Gspandl 2012], les auteurs se proposent de couvrir les fautes d'environnement et de la couche matérielle par le niveau décisionnel. Un système de gestion de croyance est utilisé comme représentation de l'état du système. Les erreurs provoquent des incohérences entre les observations, les consignes aux actionneurs et l'état supposé du système. Le système de gestion de croyance formule des hypothèses, et sélectionne la plus probable. Ainsi, les incohérences sont résolues et la planification prend en compte l'hypothèse.

Lussier propose un niveau décisionnel tolérant à ses propres fautes [Lussier 2007a], [Lussier 2007b]. Plusieurs planificateurs, diversifiés, sont implémentés. Un composant en assure la coordination et réalise une vérification des plans générés vis-à-vis de propriétés issues de l'expertise. Ce composant, qui est la clé de voûte de la tolérance aux fautes, n'est pas tolérant à ses propres fautes mais il est suffisamment simple pour que l'élimination des fautes puisse être suffisante. Enfin, il est indépendant des planificateurs qu'il contrôle. Même si la sécurité est un des objectifs affichés, l'évaluation de l'approche ne se fait que sur des critères de fiabilité.

1.4.2 Mécanismes dans le niveau exécutif

Dans les travaux précédents le recouvrement se fait implicitement par l'enrichissement des données de la planification. Or le planificateur ne peut pas régler toutes les situations, notamment lorsque le robot ne dispose pas, ou plus, des ressources suffisantes. Une solution est alors d'adapter le niveau d'autonomie du système à son état. Dans l'approche proposée par [Durand 2010], les fautes d'environnement et de capteurs sont couvertes par le niveau exécutif. Lors d'une erreur, celui-ci exécute la fonction concernée en mode dégradé, choisit une autre fonction pour remplir la même tâche, ou réduit le niveau d'autonomie du système en demandant à un opérateur d'intervenir en télé-opération. Ceci n'est évidemment possible que pour certaines applications. D'autre part, l'adaptation de l'autonomie permet surtout de finir la mission mais ne résout pas les éventuels problèmes de sécurité, plus immédiats. La tolérance aux fautes est, en effet, souvent vue comme un moyen de fiabilité avant d'être un moyen de sécurité. L'approche est étendue dans [Crestani 2015], notamment en l'inscrivant dans le processus de développement du système. Les risques sont d'abord identifiés par une analyse Amdec. La détection des erreurs est faite par des moniteurs, dont sont dotés chaque module du niveau fonctionnel.

Dans [Py 2004a] et [Py 2004b], un niveau exécutif pour la tolérance aux fautes est développé. Les propriétés de base du composant sont :

— l'observabilité, ou la capacité à observer les évènements menant à des états incohérents,

— la contrôlabilité, ou la capacité à éviter les états incohérents,

— un fonctionnement synchrone et un temps d'exécution borné,

— la vérifiabilité.

L'observation des séquences d'évènements issus des modules fonctionnels permet au niveau exécutif de maintenir à jour une représentation de l'état du système et de l'environnement (sous forme de graphe). Des propriétés issues des spécifications ou de l'expertise sont vérifiées en permanence. Les moyens de contrôle sont le blocage de requêtes et l'interruption de l'exécution d'un module. Les étapes du plan incohérentes avec l'état du graphe ne sont donc pas exécutées, ce qui couvre les fautes du plan et certaines situation adverses. Les requêtes entre les modules fonctionnels sont également filtrées. Cependant, comme le niveau exécutif dépend du niveau fonctionnel pour l'observation des états du système, il ne peut pas couvrir toutes les fautes du niveau fonctionnel. Pour résoudre le problème de la vérifiabilité, le contrôleur a été synthétisé à partir d'un modèle formel du niveau fonctionnel dans [Bensalem 2009].

1.4.3 Mécanismes séparés de la commande principale

La défense en profondeur est un principe très utilisé en tolérance aux fautes. Elle vise à introduire à chaque niveau de décomposition du système un mécanisme de tolérance. Directement attaché à un composant, le mécanisme peut détecter et couvrir finement les erreurs, ce qui évite toute propagation. En revanche, le composant et son moniteur peuvent avoir des fautes de mode commun, notamment lorsqu'ils partagent le même support d'exécution. Par exemple, dans [Tomatis 2003], la sécurité d'un robot guide de musée est gérée à plusieurs niveaux par : 1) le traitement des exceptions par le système d'exploitation, 2) un moniteur de redondance logicielle et 3) un moniteur de redondance matérielle. Le moniteur logiciel surveille les tâches logicielles critiques, comme le contrôle de vitesse. Il peut intervenir en relançant une tâche, en stoppant le robot et en déclenchant des alarmes. Comme ce moniteur s'exécute sur le même matériel que le contrôle fonctionnel, un autre moniteur, indépendant matériellement, a été ajouté. Ce moniteur correspond au dernier niveau de défense, séparé de la commande principale. Son observation est partielle, ses interventions sont limitées et il ne peut pas intervenir finement. En revanche, il n'a aucune faute de mode commun avec le reste du système. Ici, le moniteur séparé intervient en coupant l'alimentation des moteurs lorsque le pare-choc est enfoncé ou que la vitesse dépasse un certain seuil. L'approche suivie est totalement *ad hoc* et la liste de tâches critiques initiale semble résulter de la seule expertise. Au cours de l'expérience, des tâches, d'abord considérées comme non-critiques, sont ajoutées à la liste. Onze guides robotiques ont été déployés pendant cinq mois d'utilisation, sans aucun accident.

Dans [Woodman 2012] et [Woodman 2013], un moniteur s'exécute en parallèle de la commande principale et applique des règles de sécurité qui prescrivent des interventions lorsque leurs conditions ne sont pas vérifiées. Le moniteur est lui-même constitué de plusieurs fils d'exécution qui, en parallèle, mettent en vigueur une règle chacun. En cas d'observations trop incertaines ou lorsque les règles le prescrivent, les requêtes aux actionneurs sont filtrées, ou le robot est stoppé. Pour ce faire, le moniteur intervient sur les interfaces des actionneurs. Bien que le moniteur ait son propre modèle de l'état du système, il n'est pas complètement indépendant du contrôle principal pour l'observation et donc pour la mise à jour de son modèle. Afin d'améliorer la disponibilité du système, le moniteur communique au contrôleur principal les restrictions en vigueur. La méthode est appliquée en simulation sur un robot mobile dont la tâche est de contrôler la qualité et de trier des pièces. Une des originalité de l'approche réside dans le fait qu'elle comprend l'identification des règles de sécurité.

1.4.4 Identification et expression des règles de sécurité

La sécurité se traduit dans le contexte de chaque système par des propriétés qui doivent toujours être vraies : les *invariants de sécurité*. Afin que ces invariants restent continuellement vrais, des *mesures de sécurité* sont prises, qui représentent la façon dont la sécurité est assurée. Elles régissent ce que le système et l'environnement doivent faire ou satisfaire, par exemple des règles d'opération, ou des *règles de sécurité* appliquées par le moniteur séparé de sécurité.

L'analyse de risque vise à déterminer systématiquement les invariants de sécurité. Des mesures de sécurité peuvent être proposées pour répondre à chaque invariant pendant l'analyse. Cependant, le passage des invariants aux mesures est informel et expert. Or, la complexité des systèmes, ou la volonté de les certifier, tendent à rendre cette étape plus traçable et formelle.

Un des traits les plus originaux des travaux de Woodman est de lier directement et explicitement l'analyse de risque aux règles exécutées par le moniteur. La technique d'analyse (l'Hazop-SHARD) est mentionnée et réalisée sur le cas d'étude. Chaque invariant issu de l'analyse donne lieu à une ou plusieurs règles. Cependant, le passage des invariants aux règles se fait de manière intuitive et informelle. Les règles sont exprimées sous la forme fixes «Si ... Alors ...». La condition est formulée avec des observations de capteurs et la conclusion est la restriction de la commande d'un ou plusieurs actionneurs.

Le passage des invariants aux règles est exploré dans [Mekki-Mokhtar 2012a] et [Mekki-Mokhtar 2012b]. Le terme «invariant» est restreint aux propriété formalisables par des prédicats sur des variables observables par le moniteur. Chaque invariant est représenté par un graphe, comme illustré par la figure 1.4. Chaque état est défini par un ensemble de valeurs pour les variables observables. Les valuations violant l'invariant sont réunies dans un état dit catastrophique. Sous certaines conditions, une marge de sécurité est prise, ce qui se traduit par la partition en deux d'un état. Ce graphe permet de visualiser les différents chemins que peut emprun-

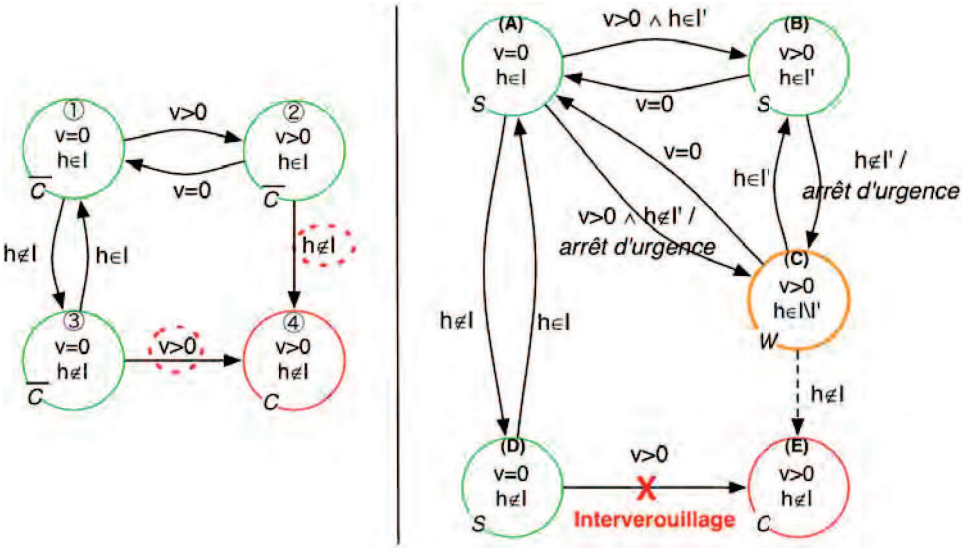


FIGURE 1.4 – Traitement de l'invariant «Les poignées doivent être à la bonne hauteur». L'invariant est formalisé avec les variables observables v, la vitesse, et h la hauteur des poignées. Sur le graphe de gauche, les transitions menant dans l'état catastrophique sont entourées en pointillées. Pour celle correspondant à un changement de hauteur, une marge peut être prise, ce qui explique le partitionnement de l'état 2 pour donner les états (B) et (C). Une intervention, l'arrêt d'urgence, est déclenchée lors de l'entrée dans l'état de marge. La deuxième transition vers l'état catastrophique est traitée par un interverrouillage, qui, en bloquant les roues, empêche le démarrage.

ter le système pour violer l'invariant et donc les interventions à déclencher pour l'en empêcher. A partir de la liste des interventions déclenchables par le moniteur, des interventions sont choisies. Le tracé du graphe est manuel, tout comme l'assignation d'une intervention. L'association d'une transition et d'une intervention permet de définir une règle de sécurité de la forme «Si ... Alors ...», destinée à être vérifiée par un moniteur séparé. Nous reviendrons sur cette approche dans le chapitre 2.

Le problème des interactions entre mesures de sécurité est soulevé dans [Alexander 2009b] et [Black 2009]. En effet des règles de sécurité peuvent entrer en conflit conduisant à l'aggravation du risque. Par exemple, une voiture autonome aura probablement comme règles de sécurité de 1) freiner lorsqu'un obstacle est détecté devant la voiture, 2) accélérer lorsque la distance avec le véhicule de derrière est insuffisante. Dans une situation où les deux règles s'appliqueraient, le comportement de la voiture est peu prévisible, conduisant à un danger ajouté par le conflit de deux règles.

1.5 Bilan

Nous nous proposons dans cette section de reprendre et discuter les points saillants de l'état de l'art. La tolérance aux fautes est un moyen privilégié pour la sécurité-innocuité des systèmes autonomes, étant donnés 1) la complexité des

systèmes, qui ne peuvent pas être considérés sans fautes résiduelles et 2) l'environnement ouvert et les multiples interactions attendues. La tolérance aux fautes peut en particulier être implémentée dans un composant que nous avons appelé *moniteur séparé de sécurité*.

1.5.1 Du moniteur de sécurité et de ses spécifications

Le moniteur doit être fiable comme doivent l'être les ressources qu'il utilise : les capteurs, les actionneurs et leurs interfaces, le matériel sur lequel s'exécute le moniteur. Leur implémentation nécessite donc des mesures spécifiques :

— Des techniques caractéristiques de la tolérance aux fautes : indépendance vis-à-vis de la commande principale pour tolérer les fautes du systèmes, redondance et surveillance temporelle de l'exécution pour tolérer les fautes du moniteur.

— La prévention des fautes lors de la conception du moniteur, c'est-à-dire un procédé de conception rigoureux voire formel comme la synthèse.

— L'élimination des fautes, incluant le test et la vérification. Le moniteur doit être suffisamment simple pour que l'élimination des fautes ne laisse pas de fautes résiduelles.

Les méthodes formelles sont donc un moyen privilégié dans la conception d'un moniteur de sécurité. Par exemple, un moniteur est synthétisé grâce à la vérification en ligne dans [Pike 2012].

En plus des spécifications techniques concrétisant sa fiabilité, le moniteur est régi par des spécifications fonctionnelles haut-niveau qui déterminent le comportement des sorties en fonction des entrées en s'abstrayant du fonctionnement interne. Ces spécifications sont appelées *règles de sécurité*. Des travaux présentés se dégage une forme récurrente de règles. Les entrées sont des observations du système et de l'environnement. Elles correspondent à des capteurs acquérant des grandeurs physiques ou à des observations de l'état interne. Les observations sont mémorisées sous forme d'un modèle de l'état du système. Les sorties sont liées à la gestion des tâches logicielles, à l'inhibition des actionneurs ou à la commande d'actionnement (par exemple, le cas de l'arrêt dans [Tomatis 2003]). Les règles sont composées d'une condition, formulée sur les observations, et d'une intervention. La condition régit l'application de l'intervention.

Nous pensons que l'implémentation fiable d'un moniteur n'est pas suffisante. Son efficacité repose aussi dans les règles de sécurité qu'il exécute. On a vu dans [Tomatis 2003] qu'un ensemble de règles de sécurité déterminé empiriquement n'est pas toujours complet. Aucun argument ne peut être avancé pour évaluer le niveau de confiance dans sa complétude. La démarche de Woodman, qui lie explicitement les résultats de l'analyse de risque à la partie observation des règles, permet au contraire de justifier d'une méthode systématique. Elle ne justifie cependant pas que les interventions choisies soient suffisantes pour éviter les dangers.

1.5.2 Des interventions

Généralement, peu d'intérêt est porté aux moyens d'intervention des moniteurs. Les systèmes autonomes héritent en cela de la double tradition du logiciel et des systèmes critiques classiques. La vérification en ligne, issue du logiciel, ne gère a priori que la levée des erreurs mais pas leur traitement, considéré comme un problème *ad hoc* [Goodloe 2010]. Dans les systèmes critiques (avion, train, centrale nucléaire, etc), des opérateurs humains sont disponibles, en dernier recours. La gestion des situations dangereuses et exceptionnelles leur incombe, et est même devenue leur tâche principale. Les moniteurs sont alors chargés de déclencher des alarmes, l'intervention directe étant décidée par l'opérateur [Papadopoulos 2001]. Dans les systèmes autonomes au contraire, le déclenchement d'une intervention ne peut pas dépendre d'un opérateur. L'intervention doit donc être choisie *a priori* au même titre que les conditions correspondant aux alarmes.

Dans les exemples donnés, les moniteurs possèdent souvent des moyens d'intervention peu diversifiés. Ceci peut se justifier par le niveau de fiabilité requis pour chaque intervention. Cependant, appliquer la même intervention (en général l'arrêt) dans toutes situations dangereuses n'est pas adaptée lorsque ces situations sont très diverses. L'étude [Malm 2010] porte sur les accidents mettant en cause des robots industriels en Finlande, sur une vingtaine d'années. Elle conclut que l'écrasement est la situation la plus dangereuse et préconise dans ce cas un mouvement de recul au lieu d'un arrêt. L'intervention doit donc être adapté à la situation, c'est-à-dire que la partie intervention d'une règle de sécurité doit être confrontée aux cas de violation de la partie observation de la règle.

Pour ce faire, des méthodes formelles peuvent être utilisées. Par exemple, les auteurs de [Fotoohi 2011] utilisent la synthèse de superviseur ([Wonham 2005], [Ramadge 1987]) pour vérifier les règles de sécurité d'un fauteuil roulant semi-autonome. Sur un modèle états-transitions du système, ce formalisme requiert de définir les transitions comme contrôlables ou non. Une transition contrôlable correspond à la capacité du moniteur d'empêcher le franchissement de la transition. On notera également que la vérification semble très adaptée pour examiner le problème des conflits entre les règles de sécurité appliquant des interventions incompatibles.

1.5.3 De l'anticipation du danger et de la polyvalence

Une approche intéressante dans [Ertle 2010] est de propager le risque dans les états précédant les états dangereux pour anticiper le danger. Anticiper est capital pour pouvoir éviter un danger. Cela implique de prendre une marge : à partir de quand le système est-il «trop proche» d'un danger (mais pourtant pas encore dans un état dangereux) ? Dans [Sha 2001], la marge est définie par les capacités du moniteur à éviter que les invariants de sécurité ne soit violés. En effet, le contrôle que le moniteur peut exercer sur le système est limité, même si des interventions diversifiées sont disponibles. Les interventions de sécurité

contraignent le comportement du système, l'empêchant de remplir toutes ses tâches normalement (par exemple, avec des modes dégradés dans [Durand 2010]). Une large marge, a priori le choix le plus prudent, empêche le système de fonctionner librement sur un grand espace d'état. Il y a donc un compromis à trouver entre la sécurité et la polyvalence. La question n'est souvent qu'effleurée dans la littérature, sauf dans les cas de synthèse.

— Dans le cadre de la vérification en ligne [Pike 2012], la *fonctionnalité* est définie comme le fait que le moniteur ne puisse pas intervenir si aucune spécification n'a été violée.

— La synthèse de superviseur génère un moniteur optimal dans le sens où il permet le comportement sûr le plus étendu possible.

1.6 Conclusion

Les systèmes étudiés, par leur nature autonome et critique, requièrent des mesures de tolérance aux fautes pour assurer la sécurité-innocuité. Ces mesures comprennent la mise en place de mécanismes à l'intérieur ou à l'extérieur de la commande principale. Peu de travaux ont été menés sur le sujet. Certains sont très *ad hoc*, d'autres ne couvrent qu'une partie du système et aucun ne traite le problème de bout en bout. Pour ce faire, il faut dans un premier temps s'intéresser au moniteur séparé, ultime recours en cas de défaillance catastrophique. Dans le processus de développement, un lien clair entre les résultats de l'analyse de risque et les règles de sécurité fait défaut. Au niveau du système, le rebouclage des sorties du moniteur sur l'état du système n'est pas considéré explicitement, comme l'impact du moniteur sur la polyvalence du système. Une méthode générique et générale manque et cette thèse vise à combler ce manque en élaborant une méthode de spécification des règles de sécurité.

Ce qu'il faut retenir :

- Les caractéristiques des systèmes autonomes critiques sont : la complexité, un environnement ouvert et peu structuré, des interactions variées avec les êtres humains.
- La tolérance aux fautes est un moyen privilégié pour assurer la sécurité des systèmes autonomes.
- En présence de fautes, la sécurité repose sur le moniteur de sécurité. La confiance qu'on a dans le moniteur est justifiée par la prévention et l'élimination de ses propres fautes, notamment par l'utilisation de méthodes formelles.
- Les règles qui assurent la sécurité doivent être issues de l'analyse de risque.
- Les règles qui assurent la sécurité doivent tenir compte des interventions possibles et de la polyvalence du système.

Chapitre 2

Problématique : la spécification des règles de sécurité

Sommaire

Bien que la sécurité-innocuité soit un défi important pour les systèmes autonomes, ce sujet reste peu abordé par la communauté de recherche en sûreté de fonctionnement. Étant données les caractéristiques des systèmes autonomes, la tolérance aux fautes a été identifiée dans le chapitre 1 comme un moyen indispensable pour assurer leur sécurité-innocuité, et une technique particulière a été introduite : le «moniteur de sécurité». Ce composant, séparé de la commande principale, doit être spécifié et implémenté en utilisant des techniques de prévention et d'élimination des fautes.

Nous nous intéressons plus particulièrement aux spécifications fonctionnelles haut-niveau du moniteur, ou «règles de sécurité». Ces règles régissent le comportement global du moniteur ou, d'un autre point de vue, sont appliquées en ligne par le moniteur.

L'état de l'art nous permet d'identifier les informations nécessaires à l'élaboration des règles de sécurité et leurs propriétés désirables. La figure 2.1 résume une première approche «boîte noire» de cette problématique. D'une part, une méthode de spécification des règles doit s'appuyer sur une analyse de risque, comme suggéré par [Ertle 2010] et plus encore par [Woodman 2013]. D'autre part, les règles doivent tenir compte des capacités du moniteur à observer et intervenir. Enfin, les interventions de sécurité ont la capacité à empêcher le système de remplir toute tâche. Le moniteur doit donc non seulement assurer la sécurité mais aussi permettre au système de remplir ses tâches.

En terme de moyens, les méthodes formelles sont à privilégier. L'élaboration des règles doit donc reposer sur un modèle formel, ce qui permet d'expliciter toutes les

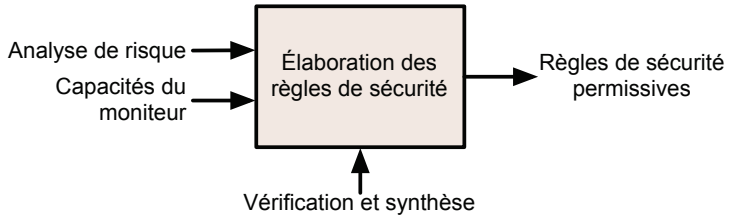


FIGURE 2.1 – Vision Entrées/Sorties de la problématique

hypothèses faites sur l'environnement et le système. Parmi les méthodes formelles, la synthèse permet d'obtenir directement des règles sans processus d'essai-erreur lié à l'élimination des règles non-satisfaisantes. Nous chercherons donc, autant que faire se peut, à synthétiser les règles de sécurité.

Ce chapitre a pour but de donner les hypothèses et les définitions sur lesquelles reposent nos travaux en s'abstrayant de toute formalisation qui constituerait déjà un choix précis de modélisation. Il s'agit de poser la problématique et les premiers choix afin de dégager le squelette d'une méthode que nous développerons dans les chapitres suivants. Nos travaux font suite à ceux de Mekki-Mokhtar [Mekki-Mokhtar 2012a], dont nous reprenons l'idée d'exploiter des marges de sécurité. Nous adaptons ses définitions et ajoutons les notions de permissivité, de stratégie et de synthèse. Nous présentons dans le paragraphe 2.1 le moniteur de sécurité tel que nous le considérons, et dans le paragraphe 2.2 les propriétés du moniteur qui constituent notre objectif. Cet objectif sera rempli par la détermination des règles et des stratégies de sécurité, introduites à la section 2.3. Le paragraphe 2.4 explicite l'approche choisie, par état d'alerte, ou par marge. Enfin, la synthèse est discutée au paragraphe 2.5. À l'issue de ce chapitre, une méthode d'élaboration des règles de sécurité est ébauchée par la définition d'étapes et de leur enchaînement.

2.1 Le moniteur de sécurité

S'appuyant sur les exemples de la littérature présentés au paragraphe 1.4.3, nous définissions le moniteur de sécurité et ses attributs. Au contraire de cette littérature et comme préconisé dans le paragraphe 1.5.2, nous considérons les interventions dans toute leurs généricité.

Le moniteur de sécurité est un dispositif de sécurité qui a des moyens d'observations et d'interventions. Son comportement est régi par des règles de sécurité.

Les observations sont, pour le moniteur, des variables qu'il reçoit en entrée. Elles concernent l'état du système ou de son environnement.
Exemples : la vitesse d'un bras de robot, l'enfoncement d'un pare-choc, la température, l'état fermé ou ouvert d'une pince.

Les interventions sont les sorties du moniteur. Ce sont les moyens que le moniteur a pour influer sur le comportement du système.

Exemples : le freinage, l'ouverture du circuit d'alimentation d'un moteur, l'alimentation d'un radiateur, le blocage d'une pince.

La chaîne de sécurité, constituée du moniteur et de ce dont il dépend pour ses observations, ses interventions et son exécution, est supposée *sans faute*. En particulier, il ne doit y avoir aucune faute de mode commun entre la chaîne de sécurité et la commande principale. Cette hypothèse peut être justifiée entre autres par :

— la simplicité du moniteur (relativement à la commande principale),

— l'utilisation de la redondance,

— des processus de spécification et de développement rigoureux,

— une certaine indépendance entre la chaîne de sécurité et la commande principale. Ainsi la norme [IEC61508-7 2010] définit au paragraphe C.3.4 un *dispositif de surveillance diversifiée* :

> «Un dispositif de surveillance diversifiée est un système de surveillance externe qui est mis en œuvre à partir d'un ordinateur indépendant selon une spécification différente. Ce dispositif a pour seul objectif d'assurer que l'ordinateur principal exécute des opérations sûres mais pas nécessairement correctes. [...] Il évite par ailleurs tout état d'insécurité du système. [...] »

Cette définition souligne l'indépendance des spécifications de la commande principale et du moniteur. En effet, les deux entités n'ont pas les mêmes objectifs : la commande principale est responsable de l'accomplissement des tâches, voire de l'optimalité du comportement du système, tandis que le moniteur est seulement responsable de la sécurité. L'accent est également mis sur l'indépendance des supports d'exécution du moniteur et de la commande principale.

Dans ces travaux, nous ne traiterons pas plus avant des moyens de justifier l'hypothèse que la chaîne de sécurité est fiable. Nous nous concentrerons sur les spécifications fonctionnelles du moniteur.

Les exigences de fiabilité de la chaîne de sécurité amènent à avoir peu d'interventions et d'observations possibles pour le moniteur. En particulier, certaines classes d'observations ne peuvent pas être utilisées par le moniteur. Parmi les données issues des capteurs, la vision, par exemple, nécessite des ressources de calcul coûteuses à redonder, des algorithmes difficiles à vérifier et dépend de la luminosité de l'environnement. Les états internes du système, tels que les modes de raideur ou les plans, ne sont pas des observations utilisables, car ce sont des informations qui ne sont pertinentes qu'au sein de la commande. De la même manière, des interventions sur l'état de la commande ne se répercutent sur l'état du système que via des niveaux difficiles à rendre fiables. Les observations et les interventions portent donc plutôt sur les grandeurs physiques du système et pas sur la commande.

Comme le moniteur est séparé de la commande principale, les fautes ne sont pas tolérées en les détectant et en les diagnostiquant. Le moniteur observe les erreurs alors qu'elles se sont propagées au niveau physique, et ce, où que se situe la faute à l'origine de l'erreur, dans l'environnement ou l'architecture de la commande.

Même si les erreurs sont détectables avant de se propager au niveau physique, l'observation à ce niveau est suffisante car la sécurité-innocuité (au contraire de la sécurité-confidentialité) se manifeste en dernière instance au niveau physique, elle permet de couvrir les fautes quelle que soit leur nature. Ainsi, la sécurité peut être assurée sans reposer sur aucun état ou mécanisme de la commande principale.

Dans le cadre des systèmes autonomes, nous considérons que le moniteur a le pouvoir d'appliquer une intervention temporairement, au contraire d'une mise en sécurité classique qui ne peut être levée que par un opérateur. Parmi les interventions du moniteur, deux classes se dégagent : les actions, qui provoquent des changements d'état, et les inhibitions, aussi appelées interverrouillages, qui empêchent des changements d'état. Par exemple, un arrêt d'urgence est une action tandis que des inhibitions sont mises en place sur les actionneurs dans [Woodman 2012], sur les requêtes dans [Py 2004b].

La sécurité est assurée par le seul moniteur et ses moyens d'observation et d'intervention, qu'il est nécessaire de connaître pour spécifier les règles de sécurité. Or au stade de la spécification, ils peuvent ne pas encore être déterminés. Établir les règles de sécurité très tôt permet de déterminer quelles sont les observations et les interventions nécessaires pour assurer la sécurité. Ainsi, ne seront implémentés que les moyens utilisés par les règles pour assurer la sécurité.

2.2 Propriétés désirées

Les règles de sécurité exécutées par le moniteur sont destinées à être synthétisées. La synthèse est faite à partir d'un modèle qui inclut les propriétés que doivent satisfaire les règles. Elles sont de deux types : la sécurité et la permissivité.

2.2.1 Sécurité

Comme discuté dans le paragraphe 1.5.1, le lien entre l'analyse de risque et les règles de sécurité doit être explicite. Nous choisissons l'Hazop-UML comme analyse de risque car elle s'appuie sur les cas d'utilisation du système et peut donc être menée très en amont dans le développement du système. Les résultats de l'analyse sont des invariants en langage naturel. Comme nous nous intéressons aux règles de sécurité qui seront implémentées dans le moniteur, ces invariants doivent être formulés au moyen des seules variables observables par le moniteur. Cette formulation peut être difficile lorsque les variables adéquates ne sont pas observables. Il faut alors se servir d'observations disponibles, éventuellement additionnées d'hypothèses d'environnement, pour reconstruire l'observation nécessaire.

Formellement, nous considérons les invariants comme des propositions de prédicats sur les variables observables. Nous restreignons les prédicats à des comparaisons avec des seuils fixes. Cette forme est adaptée à la simplicité voulue pour le moniteur et est utilisée dans de nombreux systèmes réels. Dans notre cas, elle facilitera la modélisation du problème de synthèse des règles.

Tant que les invariants sont satisfaits, le système est dans un état acceptable ; tout état dans lequel un invariant est violé est dit *catastrophique*. Il est donc équivalent d'exprimer la sécurité comme la non-atteignabilité des états catastrophiques.

Sécurité. Le moniteur est dit sûr s'il assure la sécurité du système dans son entier. Les invariants de sécurité formalisent la sécurité du système, sans prendre en compte le moniteur. Le moniteur ne doit pas, en lui-même, poser des problèmes de sécurité, en utilisant simultanément des interventions conflictuelles.

De plus, nous faisons l'hypothèse, prudente, que la violation d'un invariant entraîne des dommages immédiats et irréversibles ; aucun recouvrement n'est possible. Il faudra donc que le moniteur intervienne avant qu'un état catastrophique soit atteint.

2.2.2 Permissivité

La raison d'être d'un système est de remplir des tâches et l'ajout du moniteur ne doit pas y faire obstacle au-delà de ce qui est nécessaire pour les besoins de la sécurité. Contrairement à la littérature sur les systèmes autonomes mais comme dans les approches formelles (voir le paragraphe 1.5.3), nous explicitons l'incidence du moniteur sur la polyvalence du système.

Permissivité. Le moniteur est dit permissif si le système équipé du moniteur peut évoluer librement dans tout son espace d'état.

À l'inverse de la sécurité qui restreint l'espace d'état, la permissivité vise à l'étendre le plus possible. Il faudra donc concilier les deux, l'idéal étant que les pertes de permissivité soient strictement nécessaires à la sécurité. En pratique, le compromis prend de multiples formes dans nos travaux, d'hypothèses d'environnement à la forme que nous fixons aux règles en passant par la vérification d'une classe plus ou moins large de propriétés du modèle.

2.3 Règles et stratégies de sécurité

Les règles de sécurité, définies au paragraphe 1.4.4, servent de spécifications fonctionnelles haut-niveau du moniteur de sécurité. Ces spécifications portent sur le comportement entrées/sorties du moniteur, en s'abstrayant de sa réalisation concrète. Le fait que le moniteur assure la sécurité et la permissivité ne dépend donc que des règles qu'il applique.

Une règle de sécurité associe une intervention à un ensemble d'états du système. L'ensemble d'états est exprimé par une condition sur les variables observables par le moniteur. Pour que la règle participe à la sécurité, cette condition doit être telle qu'elle se vérifie avant que l'invariant ne soit violé. Une règle a pour but d'empêcher le système d'emprunter certains chemins menant à un état catastrophique.

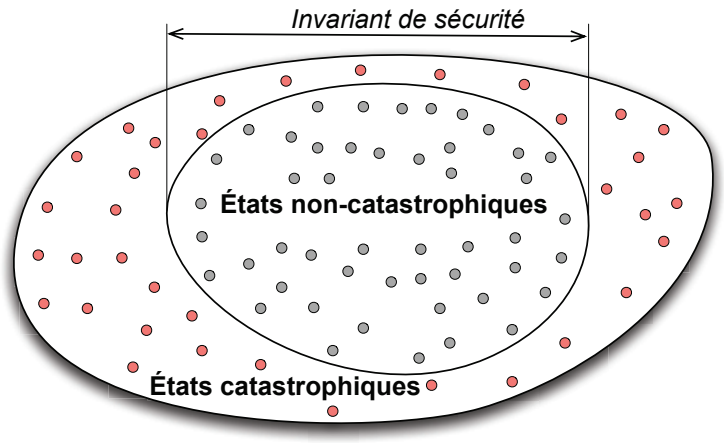


FIGURE 2.2 – Partition selon un invariant de sécurité : les états catastrophiques sont les états qui violent l'invariant.

Une stratégie de sécurité est un ensemble de règles de sécurité visant à garantir *un* invariant de sécurité, c'est-à-dire à empêcher tous les chemins menant à la violation d'un invariant. La stratégie a aussi pour but de ne pas brider la polyvalence du système, c'est-à-dire d'être permissive.

Comme dans l'ensemble de la littérature sur la tolérance aux fautes citée au paragraphe 1.4, nous choisissons de considérer des invariants de sécurité distincts au lieu de les réunir en une seule propriété de sécurité. La synthèse aura donc pour but d'élaborer une stratégie pour chaque invariant séparément. Ce choix permet de simplifier le problème en le décomposant et de passer facilement à l'échelle. En revanche, il oblige à vérifier que les interactions entre les stratégies couvrant les différents invariants ne mettent pas en danger le système : les interventions appliquées de manière concomitante ne doivent pas être conflictuelles. Cette analyse dite *des conflits* a lieu une fois que toutes les stratégies sont choisies et correspond à la recherche d'interaction conflictuelle entre interventions de sécurité. Pour que le système soit sûr, il faut donc d'une part que chaque invariant soit couvert par une stratégie de sécurité, et d'autre part que les interactions entre stratégies ne soient pas conflictuelles.

2.4 L'approche par états d'alerte

Dans le but d'élaborer des stratégies sûres et permissives, nous choisissons de considérer l'espace d'état sous la forme d'une partition.

L'invariant de sécurité définit une partition de l'espace d'état du système en *états catastrophiques* et *états non-catastrophiques*, comme schématisé sur la figure 2.2. Par exemple, si l'invariant est de ne jamais dépasser une certaine vitesse V_0, l'espace d'état du système est divisé en deux parties : d'une part, les états tels que $v \leq V_0$, et d'autre part, les états tels que $v > V_0$, qui violent l'invariant et sont donc

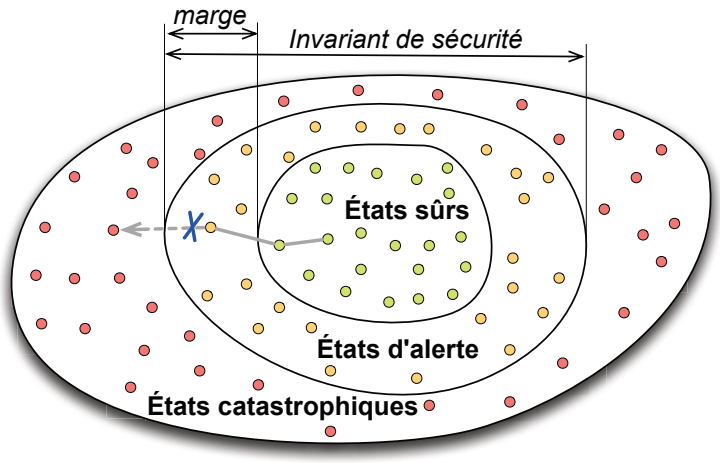


FIGURE 2.3 – Partition selon un invariant de sécurité et les marges associées. L'ensemble des états non-catastrophiques est séparé en deux parties : les états d'alerte et les états sûrs.

dits catastrophiques. Du point de vue de cet invariant particulier, la vitesse est la seule variable pertinente, peu importent les autres grandeurs définissant l'état du système.

Le respect de l'invariant implique de ne jamais atteindre d'état catastrophique. Le moniteur doit donc intervenir avant que le système soit dans un état catastrophique. En d'autres termes, les états auxquels les règles de sécurité associent des interventions doivent être non-catastrophiques. Intuitivement, il faut freiner avant V_0 si l'on ne veut pas dépasser ce seuil.

Pour anticiper le danger, une *marge* est définie à partir de la limite de l'invariant, du côté non-catastrophique. Comme sur la figure 2.3, l'ensemble des états non-catastrophiques est donc partitionné en deux : les *états d'alerte*, proches d'un état catastrophique et les *états sûrs*. Les états d'alerte sont définis de telle sorte que tous les chemins d'un état sûr à un état catastrophique passent par au moins un état d'alerte. Sur l'exemple de la limitation de vitesse, on introduit la marge V_m. L'espace d'état est donc maintenant considéré par le biais de trois parties : $v < V_0 - V_m$ (états sûrs), $v \in [V_0 - V_m, V_0[$ (états d'alerte), et $v \geq V_0$ (états catastrophiques).

Par ailleurs, les interventions déclenchées par les règles réduisent la permissivité. Nous choisissons donc d'appliquer les interventions uniquement dans les états d'alerte. Dans l'exemple, ces choix conduisent à une seule stratégie possible : appliquer le freinage lorsque $v \in [V_0 - V_m, V_0[$. L'utilisation d'un modèle formel associé à un outil de vérification permettra de déterminer si cette stratégie est sûre et permissive.

D'une manière générale, l'espace d'état est discrétisé pour ne tenir compte que des parties pertinentes vis-à-vis de l'invariant. Les états d'alerte sont déterminés en prenant une marge sur les variables observables (L'existence théorique des marges est étudiée dans [Mekki-Mokhtar 2012a]). La valeur des marges calculée par des experts du système en fonction des caractéristiques du système et du moniteur.

Déterminer si la stratégie vérifie les propriétés de sécurité et de permissivité impose de modéliser le rebouclage des interventions sur le comportement du système.

2.5 Synthèse de stratégies

Pour spécifier un moniteur de sécurité, nous devons nous appuyer sur des méthodes d'élimination ou de prévention des fautes, de préférence formelles. Pour couvrir un invariant, nous choisissons, non pas de vérifier une stratégie déterminée manuellement, mais de synthétiser les stratégies satisfaisantes. À partir du modèle, les stratégies sûres et permissives sont automatiquement déterminées. Le choix de la stratégie à exécuter par le moniteur parmi les stratégies synthétisées est du ressort de l'utilisateur.

La synthèse est particulièrement intéressante par rapport à une simple vérification lorsque le modèle n'admet pas de stratégie sûre et permissive. De nombreuses itérations infructueuses de vérification de stratégies sont économisées. Ceci n'est vrai que si la synthèse est *complète*, c'est-à-dire qu'elle garantit que, si une solution existe, elle sera trouvée. Dans le cas où aucune solution n'existe, l'utilisateur peut directement passer à la phase de modification du modèle pour faire émerger une solution. Ces modifications se traduisent par exemple par des hypothèses de structuration de l'environnement ou des pertes de permissivité.

2.6 Conclusion

L'aboutissement de ce chapitre est de définir une méthode d'élaboration de stratégies sûres et permissives. Étant donnée la nécessaire simplicité du moniteur, nous choisissons d'élaborer les règles de sécurité lors de la conception. Elles sont exécutées par le moniteur tout au long de la vie opérationnelle du système sans être modifiées. Pour concevoir ces règles, nous disposons de la spécification des moyens d'observation et d'intervention du moniteur et des invariants de sécurité, qui sont identifiés par l'Hazop-UML puis formalisés en utilisant uniquement les variables observables. Aussi longtemps que possible, les invariants sont traités séparément les uns des autres, afin de faciliter l'élaboration et le choix des stratégies.

Les principales étapes que nous suivrons pour spécifier les stratégies de sécurité sont présentées par la figure 2.4. La première étape est la construction d'un modèle qui, vis-à-vis d'un invariant, comprend toutes les informations nécessaires à l'élaboration des stratégies et, dans la mesure du possible, seulement ces informations. La deuxième étape est la synthèse des stratégies à partir de chaque modèle, donc pour chaque invariant. Après le choix d'une stratégie pour chaque invariant, les invariants sont réunis pour vérifier que leurs stratégies, synthétisées séparément, n'appliquent pas d'interventions conflictuelles.

La méthode est détaillée, outillée et illustrée dans les chapitres suivants. Le chapitre 3 traite de la modélisation formelle, utilisée pour la construction du modèle initial, la représentation des stratégies ou l'analyse des interactions conflictuelles.

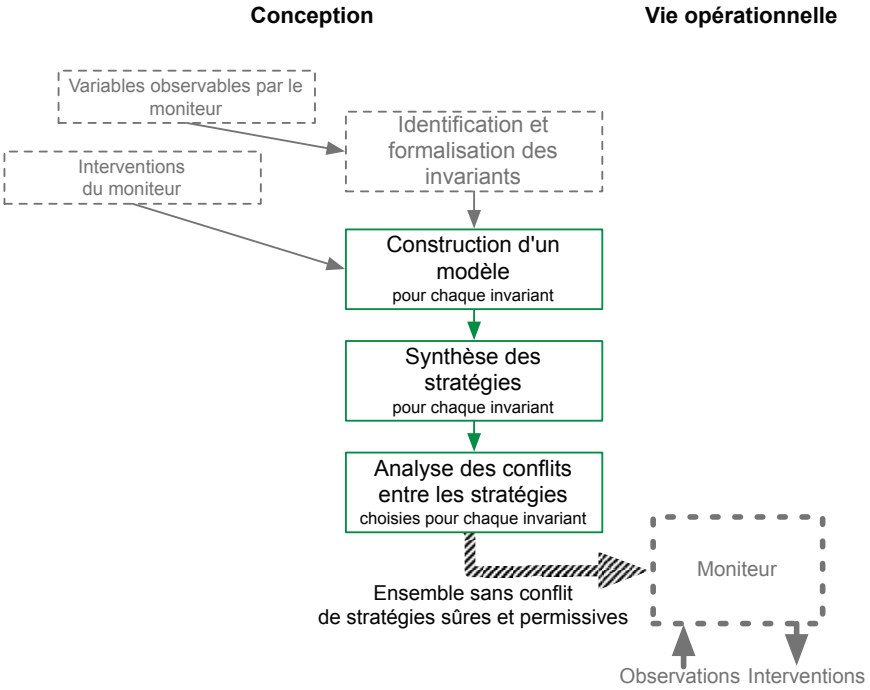


FIGURE 2.4 – Vue d'ensemble de la méthode proposée

Les mécanismes de la synthèse sont exposés dans le chapitre 4. Ces deux chapitres décrivent les outils que nous avons développés, disponibles en ligne[1]. Enfin, le chapitre 5 illustre la méthode par une étude de cas industrielle.

Ce qu'il faut retenir :

- Notre problématique est de spécifier des stratégies sûres et permissives par la synthèse.
- L'espace d'état du système est considéré par le biais d'une partition entre états catastrophiques, états d'alerte et états sûrs.
- Une stratégie associe des interventions à des états d'alerte.
- Lors de la construction du modèle et de la synthèse, les invariants sont traités séparément les uns des autres.
- La méthode proposée est schématisée par la figure 2.4.

1. `https://www.laas.fr/projects/smof/`

Chapitre 3

Modélisation

Sommaire

La synthèse des stratégies sûres et permissives nécessite en entrée un modèle qui réunisse toutes les informations nécessaires à la synthèse. Le chapitre précédent a présenté des choix qui vont structurer notre modélisation : seuls les aspects dangereux du système, c'est-à-dire les invariants, sont modélisés et ils le sont séparément les uns des autres. Nous traitons ici en détail de la modélisation des informations nécessaires à la synthèse, qui forment le modèle initial défini par l'utilisateur, de la stratégie, qu'elle soit synthétisée ou définie manuellement par l'utilisateur, et de l'analyse de conflit entre les stratégies.

La section 3.1 illustre les notions d'invariant de sécurité, de moyens d'observations et d'interventions, de marge. L'exemple présenté servira tout au long de ce chapitre. Les bases formelles nécessaires à la construction d'un modèle sont détaillées dans la section 3.2. Synthétiser les stratégies requiert de vérifier les propriétés de sécurité et de permissivité, ce qui peut être fait par des outils existants

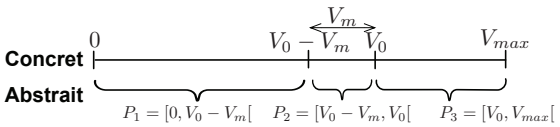


FIGURE 3.1 – Partitionnement de l'intervalle de vitesse

Concepts	Exemple
Invariant	$v < V_0 \vee r = vrai$
Variables observables	Vitesse de la plateforme v, position du bras r
Marge	Seulement sur la vitesse : V_m
Interventions	Freinage de la plate-forme, blocage du bras

TABLEAU 3.1 – Des concepts au modèle, sur l'exemple

de vérification de modèle. L'encodage du modèle est présenté dans la section 3.3, avec les outils que nous avons développés pour faciliter cette tâche à l'utilisateur. Le modèle ainsi construit constitue l'entrée de la synthèse, dont les résultats sont discutés dans la section 3.4. Une fois qu'une stratégie est choisie pour chaque modèle, c'est-à-dire que chaque invariant est couvert par une stratégie, l'analyse de conflits entre stratégies est conduite, comme exposé dans la section 3.5.

3.1 Exemple introductif

La modélisation est abordée sur un exemple, qui illustre à la fois le lien avec les étapes préliminaires et les concepts exposés dans ce chapitre (voir le tableau 3.1). Le système considéré est une plate-forme mobile équipée d'un bras robotique, la position de repli du bras est dite «position de repos». L'invariant que nous prenons pour exemple est en langage naturel, tel qu'issu de l'analyse de risque : *«Le bras doit être en position de repos lorsque la vitesse de la plate-forme est supérieure à V_0»*. Considérons deux observations pour le moniteur : la vitesse v de la plate-forme, en valeur absolue, et la position du bras, observable par une variable booléenne r (vraie lorsque le bras est en position de repos, fausse sinon). L'invariant est formalisé par $v < V_0 \vee r = vrai$.

Notre modélisation consiste à abstraire les valeurs des variables par des parties de leurs domaines de définition. La partition est faite à partir des seuils définis dans l'invariant et par les marges, ce qui permet de considérer dans l'espace d'état abstrait des états catastrophiques, des états d'alerte et des états sûrs.

La partition du domaine de définition $[0, V_{max}[$ de la vitesse est illustrée dans la figure 3.1. L'invariant de sécurité détermine la partie $[V_0, V_{max}[$. Nous partitionnons l'intervalle restant $[0, V_0[$ en deux intervalles $[0, V_0 - V_m[$ et $[V_0 - V_m, V_0[$, en introduisant la marge V_m. Finalement, le domaine de v est partitionné en trois intervalles. La position du bras est booléenne et n'admet donc aucun partitionnement autre que celui défini par l'invariant $\{\{vrai\}, \{faux\}\}$. Le produit cartésien

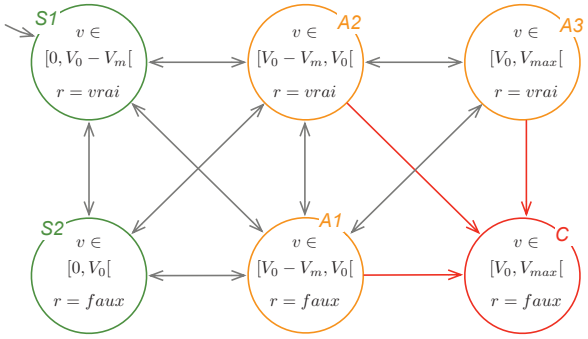


FIGURE 3.2 – Automate du comportement dont l'espace d'état abstrait est issu de la partition de l'espace d'état du système vis-à-vis de l'invariant de sécurité et des marges. Pour alléger la figure, les transitions qui bouclent directement sur un état sont omises.

Observations	Parties
Vitesse de la plateforme v	$[0, V_0 - V_m[$, $[V_0 - V_m, V_0[$, $[V_0, V_{max}[$
Position de repose du bras r	$\{vrai\}$, $\{faux\}$

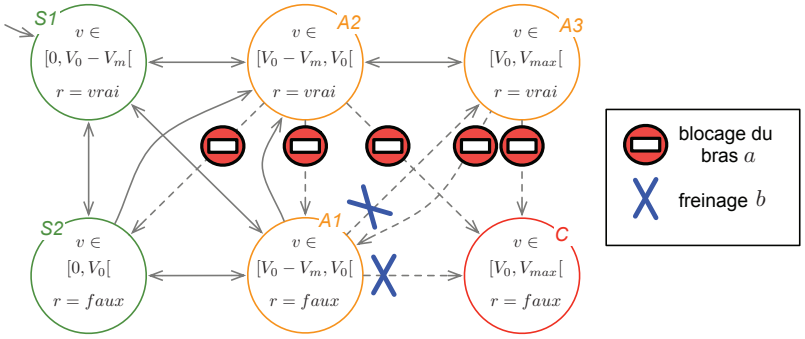
TABLEAU 3.2 – Observations du modèle introductif

Interventions	Effet	Pré-condition statique	Pré-condition séquentielle
Blocage du bras a	$r = vrai$	$r = vrai$	$vrai$
Freinage b	$v \notin [V_0, V_{max}[$	$vrai$	$v \in [0, V_0 - V_m[$

TABLEAU 3.3 – Interventions du modèle introductif

des partitions des domaines forme les états de l'automate abstrait qui représente le comportement du système, présenté figure 3.2. On considère désormais les états du système uniquement par le biais de ces états abstraits.

Les interventions considérées sont le blocage du bras en position de repos et le freinage de la plate-forme. L'effet du blocage du bras est d'empêcher le bras de se déplier, c'est-à-dire d'empêcher des évolutions d'état de $r = vrai$ à $r = faux$. Le blocage du bras n'est possible mécaniquement que lorsque le bras est en position de repos, ce qui constitue une *pré-condition* à l'application de l'intervention. Lorsque les pré-conditions sont vraies, l'effet de l'intervention doit être garanti pour toutes les conditions environnementales possibles. Il faut donc expliciter toutes les hypothèses d'environnement sous lesquelles l'effet est garanti, comme par exemple la pente maximale pour un freinage. En plus des seuils de l'invariant, des seuils de marges sont ajoutés lors de la partition. La marge V_m est dimensionnée par la latence entre le déclenchement de l'intervention au niveau du moniteur et l'effet physique du freinage. En d'autres termes, la marge doit être suffisante pour que le freinage déclenché à $v = V_0 - V_m$ (la pré-condition du freinage) garantisse $v < V_0$



États d'alerte	Interventions
$v \in [V_0 - V_m, V_0[\cup [V_0, V_{max}[\wedge r = \{vrai\}$	Blocage du bras a
$v \in [V_0 - V_m, V_0[\wedge r = \{faux\}$	Freinage b

FIGURE 3.3 – Exemple de stratégie sous sa forme d'association entre états d'alerte et intervention et ses conséquences sur l'automate. La stratégie applique le freinage lorsque le bras est déplié et la vitesse dans la marge, et le blocage du bras lorsque le bras est en position de repos et la vitesse supérieure à $V_0 - V_m$.

(l'effet du freinage). Les éléments du modèle sont repris par les tableaux 3.2 et 3.3.

Considérons la stratégie représentée en figure 3.3. L'application des interventions supprime des transitions, dont certaines ne mènent directement à l'état catastrophique. L'état catastrophique n'étant plus atteignable, la sécurité est assurée. Par ailleurs, même si plusieurs transitions non-critiques sont supprimées, la permissivité peut être jugée suffisante puisque tous les états non-catastrophiques restent accessibles.

Analysons les éléments que nous avons utilisés pour traiter cet exemple. Pour assurer la sécurité, il est nécessaire de connaître l'ensemble des chemins menant à l'état catastrophique. Ceci est réalisé d'une part par l'abstraction du comportement sous forme d'automate, d'autre part par la capacité (ici manuelle) à vérifier la non-atteignabilité de l'état catastrophique. Les autres chemins du comportement, qui ne mènent pas à l'état catastrophique, sont considérés pour la vérification de la permissivité. Enfin, la présence du moniteur est modélisée par les effets des interventions appliquées par sa stratégie, qui coupent des transitions originellement possibles. La synthèse, quoique très différente de ce processus manuel, nécessite les mêmes éléments en entrée, que l'on réunit donc dans le modèle formel que nous présentons dans la section suivante.

3.2 Structure et contenu du modèle

Pour chaque invariant de sécurité, un modèle est construit afin de synthétiser des stratégies. Le méta-modèle est donné en figure 3.4. Dans sa structure, le modèle

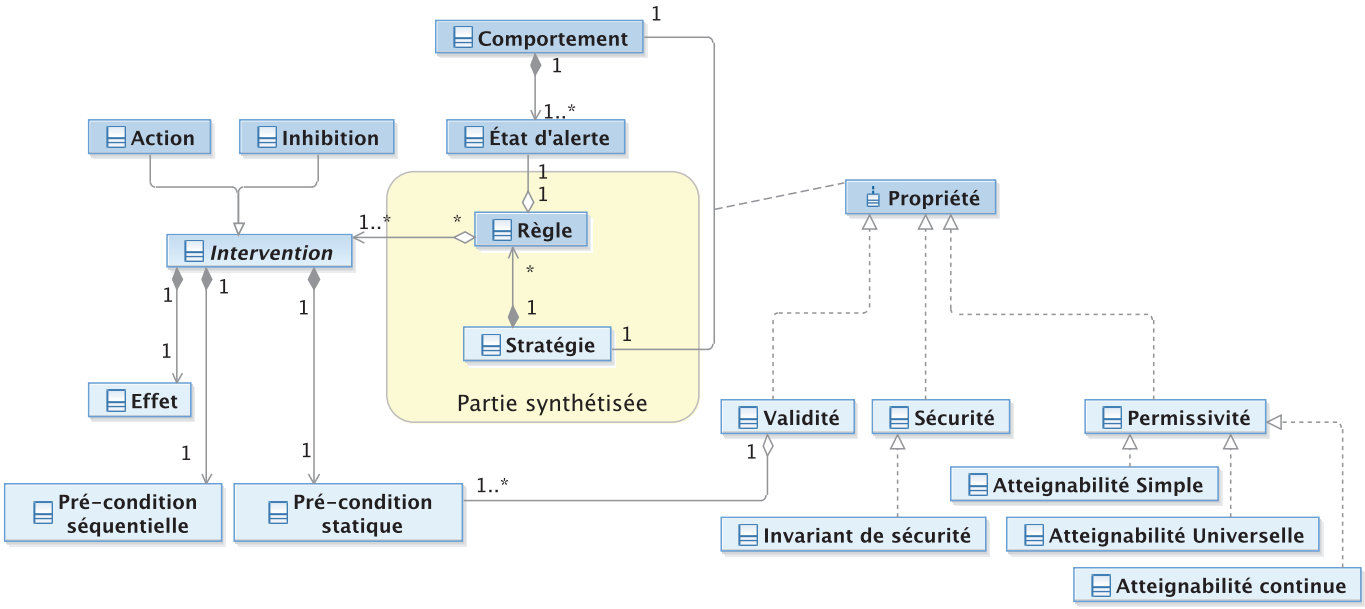


FIGURE 3.4 – Méta-modèle

tel qu'il est construit par l'utilisateur comprend trois éléments :

Le comportement est constitué de toutes les évolutions physiquement possibles de l'état du système en l'absence de moniteur.

Les interventions sont les capacités que le moniteur a pour contraindre le comportement du système.

Les propriétés sont les propriétés désirées de sécurité et de permissivité.

Le modèle est construit du point de vue du moniteur : chaque élément est modélisé à partir des variables observables par le moniteur au travers de la partition de son domaine de définition. Comme l'objectif est de synthétiser la stratégie, celle-ci est centrale dans le modèle mais laissée indéfinie par l'utilisateur. Les autres éléments, définis par l'utilisateur, sont fixes au cours du processus de synthèse. La figure 3.4 fait apparaître les relations entre les éléments. Les propriétés portent sur le comportement modifié par la stratégie. La propriété de sécurité repose sur l'invariant de sécurité tandis que la validité est une propriété auxiliaire définie à partir de pré-conditions d'interventions. La stratégie est un ensemble de règles qui associent des interventions à des états d'alerte.

La figure 3.5 schématise les relations entre les entrées de la méthode et le contenu de chaque élément qui est détaillé dans la suite.

3.2.1 Comportement

Le comportement du système vis-à-vis d'un invariant et en l'absence de moniteur est abstrait par un automate. L'idée est de ne pas considérer les états du

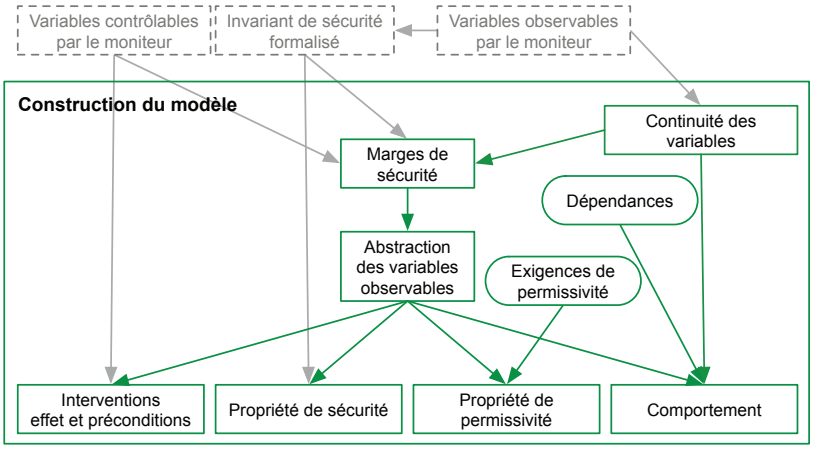


FIGURE 3.5 – Le processus de construction du modèle à partir des entrées de la méthode. Les cadres arrondis correspondent à des activités dont ne dépendent pas la sécurité. L'effort investi dans ces activités n'influence que la pertinence de la mesure de permissivité.

système, définis par des valuations de ses variables concrètes, mais d'abstraire pour chaque variable des classes d'équivalence de valeurs. Cette abstraction a pour avantage d'être finie et de tenir compte de l'invariant et des marges (et seulement de ces valeurs remarquables).

3.2.1.1 Abstraction des états

Construisons maintenant l'abstraction. Soit $\mathcal{V}$ l'ensemble non-vide des n variables observables utilisées pour formaliser l'invariant. Chaque variable $v_j \in \mathcal{V}$ a un domaine de définition D_j, doté ou non d'un ordre total. L'espace d'état concret du système est défini par la combinaison des valeurs de chaque variable, c'est-à-dire qu'un état concret c appartient à $D_1 \times ... \times D_n$. On note $c(v_j)$ la composante de l'état c correspondant à la variable v_j, c'est-à-dire la valeur de v_j dans l'état c.

Soit *Inv*, l'invariant considéré. Comme mentionné au chapitre 2, l'invariant est un prédicat dont on restreint la forme Pred_Inv à des comparaisons à des seuils :

$$\begin{aligned} \text{Pred\_Inv} &::= \text{Atome} \mid \text{Pred\_Inv} \wedge \text{Pred\_Inv} \mid \text{Pred\_Inv} \vee \text{Pred\_Inv} \\ \text{Atome} &::= v_j \; \mathit{Rel} \; \mathit{Cste} (\text{où } v_j \in \mathcal{V} \text{ prend ses valeurs dans } D_j \text{ avec } \mathit{Cste} \in D_j) \\ \mathit{Rel} &::= < \mid > \mid \leq \mid \geq \mid = \mid \neq \end{aligned}$$

Dans le cas de variable à domaine non-ordonné, les relations possibles sont réduites à l'égalité et l'inégalité. Au vu des relations admises, l'omission de l'opérateur de négation ne réduit pas l'expressivité pour l'invariant. Elle vise simplement à faciliter la définition des marges.

Pour les variables ordonnées et continues [1], des marges peuvent être ajoutées.

1. Une variable réelle est continue si ses valeurs forment au cours du temps une fonction continue. On considère qu'une variable entière est continue quand elle est seulement incrémentée ou décrémentée.

Soit l'ensemble des atomes d'invariant et de marge dans lesquels apparaît la variable v_j. Soit $\mathcal{P}_j$, une partition de D_j. Initialement $\mathcal{P}_j = \{D_j\}$.

Pour chaque atome d'invariant ou de marge v_j *Rel Cste*,

pour chaque partie $P \in \mathcal{P}_j$

si $Rel \in \{<, >, \leq, \geq\}$ et D_j ordonné, ou si $Rel \in \{=, \neq\}$ et D_j non-ordonné, alors P est partitionné en deux parties P_1, P_2 telles que $\Big(\forall e_1 \in P_1,\ (e_1\ Rel\ Cste)\Big) \wedge \Big(\forall e_2 \in P_2,\ \neg(e_2\ Rel\ Cste)\Big)$ (au sein de $\mathcal{P}_j$, l'élément P est remplacé par les deux éléments P_1 et P_2).

si $Rel \in \{=, \neq\}$ et D_j ordonné, alors P est partitionné en trois parties P_1, P_2, P_3 telles que $\Big(\forall e_1 \in P_1,\ e_1 = Cste\Big) \wedge \Big(\forall e_2 \in P_2,\ e_2 < Cste\Big) \wedge \Big(\forall e_3 \in P_3,\ e_3 > Cste\Big)$.

Supprimer les parties vides de $\mathcal{P}_j$.

FIGURE 3.6 – Procédure de partition des domaines des variables

Soit un atome d'invariant v_j *Rel Cste*, ajouter une marge revient à considérer un atome $v_j\ Rel_M\ Cste_M$, qui doit être tel que $v_j\ Rel_M\ Cste_M \Rightarrow v_j\ Rel\ Cste$. Par exemple, l'atome d'invariant de la vitesse $v < V_0$ a donné lieu à un atome de marge $v < V_0 - V_m$ plus restrictif. On note $sat(c, \varphi)$ lorsque l'état c satisfait le prédicat φ constitué d'atomes de marge ou d'invariant combinés par les opérateurs de logique propositionnelle.

Définissons maintenant des classes d'équivalence de valeurs vis-à-vis des atomes d'invariant et de marge en utilisant la procédure donnée en figure 3.6. Grâce à la forme simple des atomes, chaque atome v_j *Rel Cste* permet de partitionner D_j en deux ou trois parties. Dans le cas d'un atome $v_j < Seuil$, le seuil de l'atome définit deux intervalles : en-dessous et au-dessus du seuil. On applique itérativement cette opération pour chaque atome sur chaque partie résultant des précédentes partitions pour obtenir finalement la partition $\mathcal{P}_j$ de D_j. Par exemple, les deux atomes $v < V_0 - V_m$ et $v < V_0$ définissent pour $D_v = [0, V_{max}[$ la partition $\mathcal{P}_v = \{[0, V_0 - V_m[, [V_0 - V_m, V_0[, [V_0, V_{max}[\}$. Les partitions $\mathcal{P}_j$ issues de la procédure donnée en figure 3.6 ont les propriétés suivantes par construction :

Ordre Si le domaine D_j est doté d'une relation d'ordre total, la forme des atomes qui le partitionnent permet d'étendre cet ordre total à $\mathcal{P}_j$. Définissons l'ordre $\prec$ dans $\mathcal{P}_j$. Soient $P_1, P_2 \in \mathcal{P}_j$. On a $P_1 \prec P_2$ si et seulement si $\forall e_1 \in P_1, e_2 \in P_2,\ e_1 < e_2$.

Contigüité Si le domaine D_j est doté d'une relation d'ordre total, on définit la contigüité dans $\mathcal{P}_j$. Soient $P_1, P_2 \in \mathcal{P}_j$, avec $P_1 \prec P_2$. Les parties P_1 et P_2 sont contigües si et seulement si $\nexists P \in \mathcal{P}_j,\ P_1 \prec P \prec P_2$. Deux parties sont contigües si l'une est successeur immédiat de l'autre.

Partition Comme $\mathcal{P}_j$ est une partition de D_j, la composante $c(v_j)$ d'un état c appartiennent à un unique élément de $\mathcal{P}_j$ et l'état c appartient à un unique élément de $\mathcal{P}_1 \times ... \times \mathcal{P}_n$.

Équivalence vis-à-vis des atomes et de l'invariant Soit un atome de marge ou d'invariant φ, soient deux états concrets c, c' appartenant à un même élément de

$\mathcal{P}_1 \times ... \times \mathcal{P}_n$. On a $sat(c, \varphi) \iff sat(c', \varphi)$, c'est-à-dire que les états satisfont tous les deux l'atome ou le falsifie tous les deux. L'équivalence vis-à-vis de l'invariant en découle directement.

On abstrait un état concret c par son appartenance à un élément de $\mathcal{P}_1 \times ... \times \mathcal{P}_n$, qui est donc l'espace des états abstraits. Les valeurs dans D_j d'une variable v_j observée concrètement sont abstraites par leur appartenance aux éléments de $\mathcal{P}_j$. Pour chaque variable concrète v_j, on pose la variable abstraite $v_j^\star$ qui prend ses valeurs dans $\mathcal{P}_j$. On note $s(v_j^\star)$ la composante de l'état abstrait $s \in \mathcal{P}_1 \times ... \times \mathcal{P}_n$ correspondant à la variable abstraite $v_j^\star$, c'est-à-dire la valeur de $v_j^\star$ dans l'état s.

Les prédicats exprimés sur les variables abstraites peuvent être interprétés dans l'espace d'état abstrait. Dans un état abstrait s, la satisfaction d'un prédicat $\varphi^\star$ sur les variables abstraites est notée $sat(s, \varphi^\star)$. En particulier, on peut interpréter tout atome de marge et d'invariant v_j *Rel Cste* réécrit sur les variables abstraites par :

$$\bigvee_{P \in \mathcal{P}_{v_j RelCste}} \left(v_j^\star = P\right) \qquad \text{avec } \mathcal{P}_{v_j RelCste} = \{P \in \mathcal{P}_j | \forall e \in P,\ (e\ Rel\ Cste)\}$$

Par exemple, l'atome concret $v < V_0$ est réécrit par $v^\star = [0, V_0 - V_m[\vee v^\star = [V_0 - V_m, V_0[$. En gardant la structure logique entre les atomes d'invariant, on forme $Inv^\star$, la version abstraite de l'invariant concret Inv.

Définissons le prédicat caractéristique, noté $Carac(s)$, d'un état $s \in \mathcal{P}_1 \times ... \times \mathcal{P}_n$, vrai dans s et faux dans les autres états. Soit $s = (P_1, ..., P_n)$, on a :

$$Carac(s) = \bigwedge_{j \in 1..n} \left(v_j^\star = P_j\right)$$

3.2.1.2 Définition de l'automate

Le comportement du système en l'absence de moniteur est abstrait par l'automate $\mathcal{A} = (S, T, s_o)$ avec :

— $S \subseteq \mathcal{P}_1 \times ... \times \mathcal{P}_n$ est l'espace d'état abstrait,

— $T \subseteq S \times S$ est l'ensemble des transitions,

— $s_o \in S$ est l'unique état initial, qui doit vérifier $sat(s_o, Inv^\star)$.

En tant qu'abstraction destinée à la sécurité, l'automate doit représenter un sur-ensemble des comportements possibles du système. L'automate trivial tel que $S = \mathcal{P}_1 \times ... \times \mathcal{P}_n$ et $T = S \times S$ vérifie cette propriété. Pour garantir que tous les comportements possibles sont bien contenus dans l'automate, nous partons de l'automate trivial, duquel nous retranchons des comportements.

En premier lieu, l'hypothèse de non-recouvrement depuis un état catastrophique implique que les états catastrophiques soient des états puits, c'est-à-dire :

$$\forall s \in S,\ sat(s, \neg Inv^\star) \Rightarrow \left(\forall s' \in S,\ s' \neq s \Rightarrow (s, s') \notin T\right)$$

Deuxièmement, les variables concrètes étant des grandeurs physiques, elles sont pour la plupart continues, ce qui se traduit dans le domaine abstrait par l'impossibilité de passer d'une partie à une autre partie non-contiguë. Par exemple, sur la figure 3.1, la vitesse ne peut pas «sauter» de la partie $[0, V_0 - V_m[$ à la partie non-contigüe $[V_0, V_{max}[$. Cependant, la propriété de continuité des grandeurs physiques est altérée par l'observation à travers la chaîne de sécurité (citons par exemple l'échantillonnage). Il faut donc veiller à ce que les seuils des atomes soient suffisamment éloignés, en particulier lors du choix des seuils de marge pour garantir l'observation de valeurs intermédiaires. On supprime du comportement les transitions faisant passer une variable d'une partie à une autre partie non-contigüe. Pour toute variable $v_j \in \mathcal{V}$, pour toutes parties $P_1, P_2 \in \mathcal{P}_j$ distinctes et non-contigües :

$$\forall s, s' \in S, \left(s(v_j^\star) = P_1 \wedge s'(v_j^\star) = P_2\right) \Rightarrow (s, s') \notin T$$

Nous avons jusque là considéré que les variables sont indépendantes, comme dans le cas de la position du bras et la vitesse de la plate-forme. Lorsque ce n'est pas le cas, des transitions peuvent être supprimées pour modéliser cette dépendance. Certains types de dépendance ne sont pas exprimables dans le modèle, d'autres dépendent de la partition choisie. Illustrons ces deux difficultés en considérant la vitesse et la position d'un point sur un axe. Les variables réelles sont dépendantes mais la dérivation de la position pour obtenir la vitesse n'est pas exprimable avec des versions abstraites des variables. En revanche, le cas particulier de la permanence de la position à vitesse nulle peut être exprimé si la partition $\mathcal{P}_v$ pour la vitesse comprend la partie $\{0\}$. On peut alors exprimer cette dépendance sous une forme limitée : lorsque $v^\star = \{0\}$, la position ne change pas de partie. Lorsqu'une dépendance n'est pas exprimable, le comportement a plus de transitions qu'il ne devrait, c'est donc une sur-approximation en terme de transitions. La sécurité étant la non-atteignabilité de certains états, si elle est vérifiée sur une sur-approximation, alors elle est vraie. Au contraire, la permissivité est, par essence, une propriété d'atteignabilité. La permissivité vérifiée sur une sur-approximation ne permet donc pas de conclure. La vérification de la sécurité est toujours assurée ; la confiance à placer dans la vérification de la permissivité dépend, en revanche, de la modélisation des dépendances.

Lorsque les restrictions sur le comportement sont appliquées (non-recouvrement depuis les états catastrophiques, continuité, dépendances), on obtient l'ensemble des transitions T et l'espace d'état S réduit aux seuls états atteignables depuis s_o par T. On distingue alors trois classes d'états s dans S :

— s est un *état catastrophique* si et seulement si $sat(s, \neg Inv^\star)$.

— s est un *état d'alerte* si et seulement si $sat(s, Inv^\star) \wedge \left(\exists s' \in S,\ sat(s', \neg Inv^\star) \wedge (s, s') \in T\right)$. Il est possible d'atteindre un état catastrophique en une transition depuis un état d'alerte. L'ensemble des états d'alerte est noté S_A.

— sinon s est un *état sûr*.

Attributs	Effet E_i	Pré-condition statique $Stat_i$	Pré-condition séquentielle Seq_i
Variables admises	$v_j^\star$, $suiv\_v_j^\star$	$v_j^\star$	$v_j^\star$, $prec\_v_j^\star$

Tableau 3.4 – Variables admises dans les prédicats qui forment les attributs d'une intervention

Nous avons construit l'automate $\mathcal{A}$ représentant (au moins) l'ensemble des comportements du système vis-à-vis d'un invariant. Comme le modèle est construit pour un invariant donné, le nombre de variables pertinentes est typiquement petit, et leurs partitions comportent un petit nombre d'éléments. Ainsi la taille de l'exemple introductif est représentative de la taille des modèles réels. Dans les automates de l'étude de cas industrielle du chapitre 5, on a jusqu'à 4 variables par invariant et au maximum 3 parties par variable. En conséquence, pour synthétiser des stratégies, nous nous autoriserons des traitements qui nécessitent l'énumération des états d'alerte.

3.2.2 Interventions

Les interventions sont les moyens dont le moniteur dispose pour contraindre le comportement du système. Une intervention i est modélisée par trois prédicats sur les variables abstraites (et leurs valeurs précédentes et suivantes, cf. tableau 3.4) :

— l'*effet*, noté E_i,

— la *pré-condition statique*, notée $Stat_i$,

— la *pré-condition séquentielle*, notée Seq_i.

L'effet d'une intervention est de supprimer des transitions. Par exemple, l'effet du blocage du bras de la section 3.1 peut se modéliser par $suiv\_r^\star = r^\star$, c'est-à-dire que la valeur suivante de la variable abstraite représentant la position du bras est identique à la valeur présente (au moment où l'effet est appliqué). Exprimer un effet sur les variables abstraites nécessite donc de considérer à la fois la valeur présente $v_j^\star$ et la valeur suivante des variables abstraites $suiv\_v_j^\star$. Ainsi, le prédicat d'un effet est satisfait non pas par un état mais par une transition.

Les interventions ne sont pas toujours applicables : le blocage du bras n'est effectif que si le bras est déjà en position de repos. L'effet de l'intervention n'est possible que lorsque l'état d'application satisfait cette pré-condition. Les pré-conditions modélisent le fait qu'une intervention n'est pas toujours possible, désirable, ou que l'effet n'est pas toujours garanti. Dans le cas du blocage du bras, la pré-condition porte sur l'état d'application, c'est donc une pré-condition statique, qui s'exprime avec les seules valeurs courantes.

Toutefois, l'applicabilité d'une intervention peut aussi dépendre du chemin emprunté par le système. Par exemple, l'effet du freinage $suiv\_v^\star \neq [V_0, V_{max}[$ n'est garanti que si le freinage est déclenché sur une transition partant d'un état tel que $v^\star = [0, V_0 - V_m[$. Si le freinage est déclenché lors d'une transition arrivant d'un état tel que $v^\star = [V_0 - V_m, V_0]$, par exemple $v = V_0 - \varepsilon$, il faudrait un

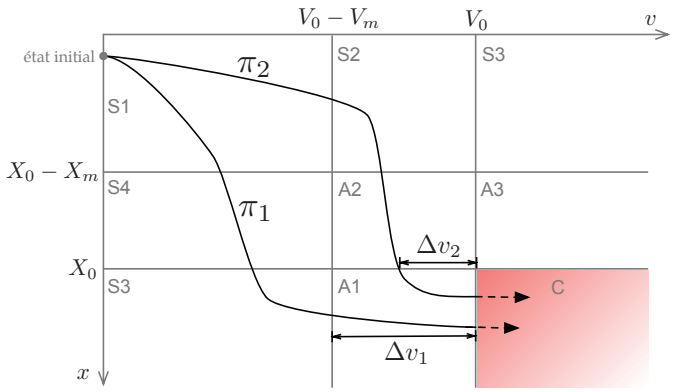


FIGURE 3.7 – Exemple d'invariant avec deux variables réelles

Attributs	Effet E_i	Pré-condition statique $Stat_i$	Pré-condition séquentielle Seq_i
Prédicats	$suiv\_v^\star \neq [V_0, V_{max}[$	*vrai*	$prec\_v^\star = [0, V_0 - V_m[$

TABLEAU 3.5 – Modélisation du freinage

freinage instantané pour empêcher qu'une valeur $V_0 + \varepsilon$ puisse être atteinte par la vitesse concrète. Nous choisissons de réduire l'historique à l'état précédant l'état d'application de l'intervention, ce qui permet déjà de modéliser les temps de latence. La pré-condition séquentielle spécifie une condition sur la transition précédant l'application. Elle nécessite donc les valeurs précédentes $prec\_v_j^\star$ des variables $v_j^\star$ pour être exprimée.

Développons maintenant l'exemple à l'aide de la figure 3.7. L'invariant considéré est modélisé avec deux variables réelles, dont la vitesse v. L'intervention de freinage n'a pas de pré-condition statique et a pour effet que la vitesse ne puisse pas être supérieure à V_0. Supposons que la stratégie associe le freinage à l'état $A1$. Sur le chemin π_1, le freinage est commandé dès qu'une vitesse concrète $v \geq V_0 - V_m$ est observée. Sur le chemin π_2, le freinage est déclenché quand $v \geq V_0 - \Delta_{v2}$. Considérons maintenant un temps de latence entre la commande du freinage et l'application effective des freins dans le système. Si la marge V_m prend en compte le temps de latence, l'effet du freinage est garanti sur le chemin π_1. Au contraire, sur π_2, une vitesse supérieure à V_0 pourra être atteinte pendant le temps de latence. En conséquence, le modèle doit prendre en compte l'état précédant l'application, ce qui est fait par l'ajout de la pré-condition séquentielle $prec\_v^\star = [0, V_0 - V_m[$. L'intervention de freinage est donc complètement modélisée par ses trois prédicats présentés dans le tableau 3.5.

Toujours dans l'exemple de la figure 3.7, supposons maintenant que la stratégie associe le freinage aux états $A1$, $A2$ et $S2$. Le long du chemin π_2, le freinage est déclenché sur franchissement du seuil $V_0 - V_m$ avec $x < X_0 - X_m$. Le temps de latence est donc pris en compte, le freinage efficace et l'état catastrophique C

inatteignable par le chemin π_2. La pré-condition séquentielle doit être vérifiée lors du seul déclenchement de l'intervention. En revanche, elle n'est pas pertinente pour maintenir une intervention. Ainsi, l'effet de l'intervention est maintenu tant que le système est en $S2$ et après le franchissement de la transition $(S2, A2)$ puis $(A2, A1)$. Pour différencier le maintien d'une intervention du commencement de son application, il faut mémoriser l'application de l'intervention au pas précédent.

Considérer des pré-conditions séquentielles nécessite donc de prendre en compte l'historique à un pas de deux informations : les valeurs précédentes des variables et l'application des interventions au pas précédent. L'espace d'état S est donc raffiné en un espace d'état S' avec une variable d'état complémentaire $prec\_v_j^\star$ par variable $v_j \in \mathcal{V}$ (initialement $prec\_v_j^\star = v_j^\star$) et une variable d'état complémentaire booléenne $prec\_App_i$ par intervention $i \in I$ (initialement fausse). Ainsi l'automate $\mathcal{A}$ est raffiné en l'automate $\mathcal{A}' = (S', T', s'_0)$. Les variables $prec\_App_i$ sont pour l'instant laissées libres : T' inclut toutes les évolutions de ces composantes d'états dans $\{vrai, faux\} \times \{vrai, faux\}$. En revanche, l'évolution des $prec\_v_j^\star$ est complètement déterminée par celles des $v_j^\star$: depuis un état s tel que $v_j^\star = P$, il n'existe que des transitions à destination d'états tels que $prec\_v_j^\star = P$.

Dans S', on peut toujours interpréter les prédicats sur les variables abstraites $v_j^\star$, et en particulier, les prédicats caractéristiques qui combinent une valeur de chaque variable $v_j \in \mathcal{V}$ (voir la fin de la section 3.2.1.1). Les prédicats caractéristiques permettent de projeter les états de S' sur S.

3.2.3 Stratégie

L'automate $\mathcal{A}$ représente le comportement du système observé par le moniteur. L'automate $\mathcal{A}'$ correspond au moniteur observateur et doté de moyens d'interventions encore inexploités. Il correspond à la simple apposition du comportement et des interventions. Des variables d'état ont été ajoutées mais leur utilisation reste potentielle. Définissons maintenant la stratégie N qui insuffle au moniteur sa réactivité, et donne lieu à un troisième automate $\mathcal{A}_N$.

Soient I l'ensemble des interventions et $\mathscr{P}(I)$ l'ensemble des parties de I. Une stratégie N associe des interventions aux états d'alerte $S_A \subset S$. On peut l'écrire comme une fonction :

$$N : S_A \to \mathscr{P}(I)$$

La stratégie N de la figure 3.3 est la fonction $N : \{A1, A2, A3\} \to \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ telle que $N(A1) = \{b\}$, $N(A2) = \{a\}$ et $N(A3) = \{a\}$.

La stratégie N définit un nouvel automate $\mathcal{A}_N = (S', T_N, s'_0)$ avec $T_N \subseteq T'$. Une stratégie est une *association* entre interventions et états d'alerte. Définissons maintenant App_i la condition d'*application* de l'intervention i par la stratégie N. La

condition App_i est vraie dans un $s \in S'$ si et seulement si s vérifie :

$$s \neq s'_o \wedge sat(s, Stat_i) \wedge \Big(sat(s, Seq_i) \vee sat(s, prec\_App_i)\Big) \wedge \Big(\exists s_A \in S_A,\ Carac(s_A) = Carac(s) \wedge i \in N(s_A)\Big)$$

Ainsi, aucune intervention n'est appliquée à l'état initial et dans les états ne respectant pas la pré-condition statique. Le raffinage de l'espace d'état de S en S' permet d'interpréter la pré-condition séquentielle et de savoir si l'intervention était précédemment appliquée. Une intervention i est appliquée dans $s \in S'$ seulement si sa pré-condition séquentielle est vraie ou qu'elle était précédemment appliquée. Enfin, une intervention n'est appliquée que lorsqu'elle est requise par la stratégie, c'est-à-dire que $s \in S'$ projeté dans S doit correspondre à un état d'alerte associé à l'intervention i. Il est à noter que l'intervention cesse d'être appliquée dès que le système sort d'un état auquel la stratégie l'associe.

Pour une stratégie donnée, la valeur de vérité de la condition d'application est déterminable dans chaque état de S'. On définit alors l'ensemble des transitions T_N en supprimant des éléments de T'. En premier lieu, la définition de App_i détermine complètement l'évolution de la variable complémentaire $prec\_App_i$, ce qui restreint T_N. Par exemple, depuis un état $s \in S'$ tel que $App_i = vrai$, il n'existe que des transitions à destination d'états tels que $prec\_App_i = vrai$. Deuxièmement, et c'est là l'objectif de la définition des automates $\mathcal{A}'$ et $\mathcal{A}_N$, on coupe des transitions grâce aux effets des interventions. Si App_i est vrai dans un état $s \in S'$, alors :

$$\forall s' \in S',\ (s, s') \in T',\ \Big(sat\Big((s, s'), \neg E_i\Big) \Rightarrow (s, s') \notin T_N\Big)$$

En projetant l'automate résultant $\mathcal{A}_N$ sur $\mathcal{A}$, on observe que les effets des interventions n'ajoutent pas de transitions : ils en suppriment ou, plus souvent, ils les rendent conditionnées par les variables complémentaires, c'est-à-dire par l'historique du chemin parcouru dans $\mathcal{A}$. En effet, comme le comportement $\mathcal{A}$ représente tout ce qui est physiquement possible, les interventions ne peuvent pas ajouter du comportement dans $\mathcal{A}$. Par ailleurs, seules les transitions sortant d'un état $s \in S$ auquel est associé une intervention i peuvent être supprimées ou conditionnées par l'effet de i.

3.2.4 Propriétés

Comme décrit par le méta-modèle de la figure 3.4, les propriétés de sécurité et de permissivité portent sur le comportement modifié par la stratégie, c'est-à-dire sur l'automate $\mathcal{A}_N$. La vérification des propriétés de sécurité et de permissivité nécessite de raisonner sur des chemins d'exécution dans $\mathcal{A}_N$. Classiquement, on va considérer $\mathcal{A}_N$ comme une structure de Kripke dont les états sont étiquetés par la satisfaction de l'invariant $Inv^\star$ et des prédicats caractéristiques. Cette structure nous permet d'interpréter des formules de logique temporelles, énonçant des

propriétés sur l'enchaînement des états le long des chemins d'exécution.

Les deux principales logiques temporelles outillées sont la LTL (*Linear Time Logic* [Pnueli 1977]) et la CTL (*Computation Tree Logic* [Clarke 1986]). Pour un premier abord de ces logiques, le lecteur peut se référer à [Schnoebelen 1999]. La LTL exprime des propriétés pour des traces au cours du temps grâce aux quantificateurs X, pour l'état suivant, G, pour tous les états de la trace, F pour l'existence d'un état dans la trace. Cependant, cette logique ne permet pas d'exprimer les différentes possibilités d'évolution au cours du temps. On utilise pour cela la CTL qui définit les quantificateurs de branche A pour toutes les branches possibles, E pour l'existence d'une branche. Un opérateur de CTL est la combinaison d'un quantificateur de temps de la LTL et d'un quantificateur de branche. En particulier, AG est l'opérateur d'invariance, EF est l'opérateur d'existence et EG l'invariance au sein d'une branche. Nous choisissons la CTL qui nous permet d'exprimer l'atteignabilité, base de la permissivité.

La propriété de sécurité s'énonce comme l'impossibilité d'atteindre un état catastrophique, c'est-à-dire $AG\ Inv^\star$.

La permissivité est la capacité du moniteur à ne pas empêcher le système d'évoluer dans son espace d'état. Cette définition ouvre de larges possibilités de modélisation. Nous choisissons de retenir l'existence de chemins entre états. Ce faisant, nous négligeons la longueur et le nombre des chemins. Un autre sacrifice, en faveur de la sécurité, consiste à ne requérir l'existence d'un chemin qu'au départ et à destination d'états non-catastrophiques.

Nous adoptons une modélisation de la permissivité décomposée par état de S et avec des degrés d'intensité. Les degrés sont obtenus par trois propriétés appliquées à un état non-catastrophique $s \in S$, de plus en plus fortes :

Atteignabilité simple de l'état s : $EF\ Carac(s)$
L'état s est atteignable depuis l'état initial.

Atteignabilité universelle de l'état s : $AG\left(Inv^\star \Rightarrow EF\ Carac(s)\right)$
L'état s est atteignable depuis chaque état non-catastrophique.

Atteignabilité continue de l'état s : $AG\left(Inv^\star \Rightarrow EF\left(Carac(s) \wedge EG\ Carac(s)\right)\right)$
L'état s est atteignable depuis chaque état non-catastrophique et le système peut y rester indéfiniment.

Certaines transitions du comportement correspondent à des changements simultanés de valeur de plusieurs variables abstraites indépendantes (correspondant à des franchissements de seuil des variables concrètes). En référence au cas à deux variables (cf. figure 3.2), ces transitions sont appelées *diagonales*. Elles sont possibles, et la vérification de la sécurité doit en tenir compte. Mais elles sont hautement improbables. Faire reposer la permissivité sur ces transitions, très difficiles à obtenir en pratique par la commande principale, n'est pas raisonnable. Les transitions diagonales ne doivent donc pas être empruntées lors de la vérification de la permissivité. Pour repérer ces transitions, on ajoute une variable complémentaire booléenne *diag*. Initialement fausse, *diag* devient (définitivement) vraie lorsqu'une

transition diagonale est franchie. Pour ignorer les transitions diagonales, les propriétés d'atteignabilité sont renforcées :

Atteignabilité simple de l'état s : $EF(\ Carac(s) \wedge \neg diag)$

L'état s est atteignable par un chemin n'empruntant aucune transition diagonale.

Atteignabilité universelle de l'état s :

$$AG\Big(\neg diag \wedge Inv^{\star} \Rightarrow EF\Big(Carac(s) \wedge \neg diag\Big)\Big)$$

On exclut les chemins qui partent d'un état tel que $diag = vrai$. L'état à atteindre doit évidemment être tel que $diag = faux$.

Atteignabilité continue de l'état s :

$$AG\Big(\neg diag \wedge Inv^{\star} \Rightarrow EF\Big(Carac(s) \wedge \neg diag \wedge EGCarac(s)\Big)\Big)$$

On transforme de même la propriété d'atteignabilité continue.

Sur la figure 3.3, on constate que l'atteignabilité universelle de l'état $A1$ est vraie. En particulier, le chemin de $A3$ à $A1$ est possible en passant par $A2$, puis $S1$, puis $S2$. Il est à noter que sans la stratégie, une transition directe est possible. La permissivité modélisée rend donc bien compte d'une propriété globale qui s'abstrait de l'existence de transitions particulières.

Par défaut, la permissivité est choisie comme la conjonction des atteignabilités universelles de tous les états non-catastrophiques. Elle peut être ajustée, en degré et par état, en fonction du comportement et des tâches du système. D'une part, le comportement, en l'absence de moniteur, peut ne pas satisfaire toutes les propriétés d'atteignabilité par défaut, notamment à cause de dépendances. Les propriétés d'atteignabilité non-satisfaites ne modélisent pas la permissivité car le moniteur ne peut pas rendre atteignable un état qui ne l'était pas en son absence. Elles ne doivent donc pas être prises en compte dans la permissivité lors de la synthèse. D'autre part, en l'absence de solutions, la permissivité sera diminuée pour arriver à un compromis avec la sécurité : l'atteignabilité de certains états peut être sacrifiée. Si l'atteignabilité continue est fausse, le système ne peut pas toujours rester dans l'état et donc y accomplir des tâches. Si l'atteignabilité universelle est fausse, le moniteur a un effet irréversible, il peut confiner le système dans un sous-ensemble d'états.

On veut vérifier qu'une stratégie N associe les interventions correctement vis-à-vis de leurs pré-conditions statiques. Pour ce faire, définissons pour chaque intervention i la variable booléenne $Asso_i$, vraie dans un état $s \in S'$ si et seulement si $\exists s_A \in S_A,\ Carac(s_A) = Carac(s) \wedge i \in N(s_A)$, c'est-à-dire que la projection de s dans S est associée à l'intervention i. La validité s'exprime alors par :

$$AG \bigwedge_{i \in I} Asso_i \Rightarrow Stat_i$$

Au sein du même modèle, il peut arriver que plusieurs interventions modélisées ne soient pas compatibles, c'est-à-dire que leur application concomitante soulève

des conflits dommageables pour la sécurité. Un conflit entre les interventions de l'ensemble I_{confl} se modélise par $\bigwedge_{i \in I_{confl}} Asso_i$. La validité est enrichie pour vérifier l'absence de conflit. Pour un modèle contenant deux interventions a et b, qui sont conflictuelles, la validité devient par exemple :

$$AG\left(\neg\left(Asso_a \wedge Asso_b\right) \wedge \left(Asso_a \Rightarrow Stat_a\right) \wedge \left(Asso_b \Rightarrow Stat_b\right)\right)$$

3.2.5 Résumé

À partir des invariants et des marges exprimés sur les variables concrètes, le comportement (du système, observé par le moniteur) est abstrait par partition des domaines de définition des variables. Le comportement est donc modélisé par un automate $\mathcal{A} = (S, T, s_o)$ dont l'espace d'état S est formé par les variables abstraites. Cet automate $\mathcal{A}$ tient compte de la continuité des variables et de leurs dépendances. Les états de S forment trois classes : les états catastrophiques, les états d'alerte (ensemble S_A) et les états sûrs.

L'ajout des interventions dans le modèle amène à considérer l'automate $\mathcal{A}' = (S', T', s'_o)$ dont l'espace d'état est raffiné par des variables d'historique. Cependant, nous raisonnerons essentiellement dans $\mathcal{A}$ grâce à la projection de S' dans S par les prédicats caractéristiques des états, c'est-à-dire les valeurs des variables d'état abstraites. Le fait que le moniteur commande physiquement l'intervention i et que le prédicat d'effet de i devienne une contrainte dans le modèle est appelé application de i. La condition d'application d'une intervention ne dépend pas seulement de la stratégie mais aussi de ses pré-conditions et de l'état précédent.

Une stratégie associe aux états d'alerte $s \in S_A$ des combinaisons d'interventions $i \in \mathcal{P}(I)$. L'automate modélisant l'application des interventions par la stratégie N est l'automate $\mathcal{A}_N = (S', T_N, s'_o)$, avec $T_N \subseteq T'$. L'application des interventions coupe des transitions.

Les propriétés de sécurité, de permissivité et de validité sont vérifiées sur $\mathcal{A}_N$, c'est-à-dire le comportement modifié par la stratégie. Cependant, lorsque le comportement et les propriétés sont fixes, ce qui est le cas pendant la synthèse, on peut qualifier directement les stratégies de sûres, permissives, valides. Les propriétés de permissivité sont formulées en différents degrés et par état de S.

3.3 Encodage du modèle

Le modèle que nous avons défini dans la section précédente est indépendant de tout outil. Or, évaluer une stratégie, donc *a fortiori* la synthétiser, requiert de vérifier les propriétés de sécurité et de permissivité. Au prix de l'encodage du modèle, de nombreux outils permettent de vérifier automatiquement des propriétés. Nous choisissons l'outil surtout en fonction de l'adéquation entre notre modélisation et le langage utilisé par l'outil. Cette adéquation détermine la complexité de la tâche d'encodage pour l'utilisateur de notre méthode.

3.3.1 Outil de vérification de modèle utilisé

Nous avons besoin d'un outil de vérification pour un modèle discret. Nous comparons ici trois des outils les plus répandus : SPIN [2] [Holzmann 1997], UPPAAL [3] [Bengtsson 1996] et NuSMV [4] [Cimatti 2002].

SPIN permet de modéliser des systèmes communicants modélisés dans le langage Promela. Le modèle est décomposé en processus paramétrables et instanciables. Un processus est constitué de procédures gardées. La procédure exécutée est choisie de manière non-déterministe parmi les procédures dont la garde est vraie. La communication entre les processus se fait par variables globales, canaux ou rendez-vous. SPIN permet de simuler le système dans une interface graphique et de vérifier des propriétés de LTL.

UPPAAL est un vérificateur d'automates temporisés. Chaque automate forme un module instanciable plusieurs fois. Le temps est modélisé par des horloges continues testées par des seuils entiers. En plus des états, des variables entières peuvent être définies. Les transitions sont gardées ; elles exécutent des fonctions (qui modifient les valeurs des variables) et permettent de synchroniser deux automates. L'interface graphique permet de dessiner et de simuler les automates mais ceux-ci peuvent également être définis textuellement. UPPAAL vérifie un sous-ensemble de la CTL : les propriétés ne comprenant qu'un seul opérateur en tête de la propriété.

NuSMV vérifie des automates. Cependant, au contraire d'UPPAAL, la structure état-transition est implicite. Des variables entières (et en général bornées) sont déclarées. Un état est une combinaison de valeurs de variables, on y fait donc référence par son prédicat caractéristique. Par défaut, une transition entre chaque paire d'états existe. La seule déclaration des variables construit donc implicitement un automate correspondant à un graphe complet. Des contraintes sont ensuite déclarées pour supprimer des états et des transitions. L'opérateur `next()` spécifie la valeur d'une variable à l'état suivant, ce qui permet de manipuler les transitions. L'automate peut être scindé en modules paramétrables et instanciables. L'interface de NuSMV se réduit à la seule console. NuSMV vérifie la CTL dans son intégralité.

La partie comportementale de notre modèle est un automate dont les états ne sont définis que comme combinaison des parties de chaque variable. Le comportement est donc très proche de l'automate généré par NuSMV lors de la déclaration des variables. Le comportement présente de nombreuses transitions, il est donc plus pratique de supprimer les transitions en surplus que de les déclarer toutes. Par ailleurs, la permissivité, sous la forme de l'atteignabilité universelle, met en jeu deux opérateurs de CTL. Elle n'est donc directement vérifiable que par NuSMV. Nous choisissons donc NuSMV (version 2.4) pour encoder et vérifier notre modèle. Dans la suite, le code et les résultats NuSMV seront mentionnés en `police machine à écrire`. Les principaux mots-clés de NuSMV sont :

2. `http://spinroot.com/spin/whatispin.html`, consulté le 7 juillet 2015
3. `http://www.uppaal.org`, consulté le 7 juillet 2015
4. `http://nusmv.fbk.eu`, consulté le 7 juillet 2015

MODULE Déclaration d'un module, qui peut être paramétré. Un modèle a obligatoirement un module nommé `main`.

VAR Déclaration d'une variable d'état de type natif ou instanciation d'un module.

DEFINE Déclaration d'une variable auxiliaire, qui ne change pas l'automate sous-jacent mais permet des raccourcis de notation. La définition d'une variable auxiliaire ne fait pas appel à `next()`, elle qualifie donc des états.

INVAR Contrainte d'invariant, c'est-à-dire suppression d'un état.

TRANS Contrainte de transition. Des transitions sont supprimées, ce qui peut rendre des états inatteignables.

CTLSPEC Définition d'une propriété de CTL à vérifier.

INVARSPEC Définition d'une propriété invariante à vérifier. Les propriétés d'invariance sont un cas particulier de propriétés exprimables en CTL pour lesquelles des algorithmes spécifiques améliorent les performances, ce qui justifie le mot-clé distinct.

La notation pour la logique propositionnelle est la suivante : `!` pour le NON, `&` pour le ET et `|` pour le OU. Les opérateurs de CTL sont transcrits directement. NuSMV vérifie les propriétés sur l'ensemble des chemins infinis du modèle. En plus de la vérification, NuSMV peut simuler le modèle. Cette fonctionnalité permet d'explorer pas à pas un chemin du modèle. Pour plus de détails, le lecteur peut se référer au manuel d'utilisation de NuSMV.

3.3.2 Construction du modèle avec NuSMV

Afin de faciliter la modélisation, nous avons défini en NuSMV un patron de modélisation. Ce modèle à trous est d'abord rempli par l'utilisateur puis complété automatiquement par un outil que nous avons développé. Nous nommons cette seconde étape la *génération*. Le modèle résultant constitue l'entrée de la synthèse de stratégie. Le modèle encodé se décompose en quatre parties :

— Une partie prédéfinie constituée des sous-modules du patron et de certains éléments de son module principal.

— Une partie à modifier par l'utilisateur, contenue dans le module principal.

— Une partie générée par l'outil de génération que nous avons développé. La partie générée tient compte d'éléments de la partie modifiée, tels que les noms de variable, mais ne requiert aucun choix de modélisation.

— La stratégie, synthétisée ou définie manuellement.

La synthèse prend en entrée un modèle sans stratégie. Cependant, le modèle n'est simulable par NuSMV que si une stratégie est ajoutée. En particulier, la stratégie vide est facile à définir manuellement et peut être utilisée pour valider le modèle par simulation avant la synthèse. Elle correspond à un moniteur passif, ne déclenchant aucune intervention.

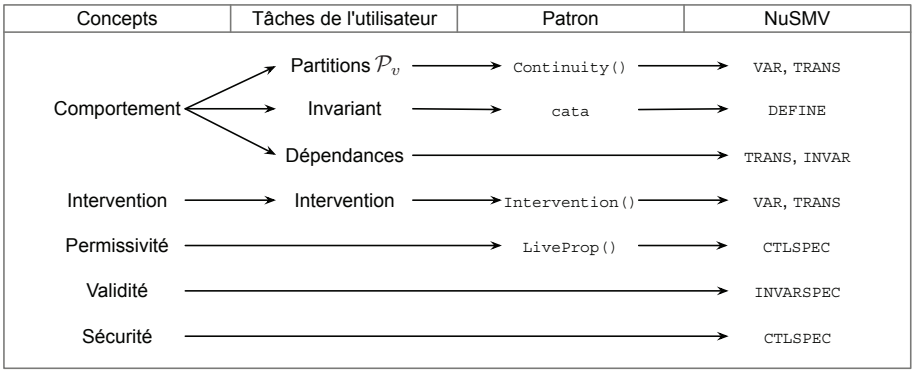


FIGURE 3.8 – Implémentation du modèle

Le patron de modélisation est présenté dans son entier en annexe A. Nous présentons ici les principaux modules et fonctionnalités du patron ainsi que l'outil de génération. Pour chaque module, une interface simplifiée est donnée en pseudo-code ; les types des arguments sont ceux attendus lors d'une bonne utilisation du patron.

La figure 3.8 illustre la correspondance entre le modèle conceptuel défini en 3.2, les tâches de l'utilisateur, le patron de modélisation et les mots-clés de NuSMV.

3.3.2.1 Encodage par l'utilisateur

Comme présenté par la figure 3.8, l'utilisateur construit le modèle en remplissant le module principal du patron pour spécifier les partitions des variables, les états catastrophiques, les dépendances et les interventions. Il s'appuie pour cela sur des éléments prédéfinis du sous-modèle.

En premier lieu, il faut construire l'espace d'état. Chaque partie de $\mathcal{P}_j$ est encodée par un entier naturel entre 0 et $card(\mathcal{P}_j) - 1$. Les variables d'état ordonnées et continues (ou les variables n'ayant que deux valeurs abstraites) sont déclarées par instanciation du module prédéfini `Continuity` (les autres variables sont déclarées directement). Deux parties contiguës sont encodées par des entiers successifs. Par exemple, la partie $[0, V_0 - V_m[$ est encodée par `0`, $[V_0 - V_m, V_0[$ par `1` et $[V_0, V_{max}[$ par `2`.

```
Continuity(cst int max_value, cst int init_value)
```

Le module `Continuity` réunit la déclaration de la variable, son initialisation et la contrainte traduisant la continuité des variables concrètes (ou la contiguïté entre les parties). Pour une variable encodée `v`, la contrainte s'exprime par :

```
TRANS next(v)= v | next(v)+1 | next(v-1)
```

Cette contrainte permet de traiter de manière générique et transparente pour l'utilisateur toutes les suppressions de transitions liées à la continuité. Une fois la déclaration des variables faite, l'automate construit implicitement par NuSMV est

l'espace d'état trivial $S = \mathcal{P}_1 \times ... \times \mathcal{P}_n$. L'ensemble des transitions tient déjà compte des contraintes de continuité.

Une variable auxiliaire (ne modifiant pas l'espace d'état) `cata` est définie par l'utilisateur comme la négation du prédicat d'invariant abstrait $Inv^\star$. L'hypothèse du non-recouvrement est exprimée sous la forme de la contrainte prédéfinie `INVAR cata -> next(cata)`. D'autres variables auxiliaires et constantes peuvent être utilisées pour faciliter la modélisation, notamment celle des dépendances. Étant spécifiques à chaque modèle, les dépendances sont directement déclarées par l'utilisateur au moyen des mots-clés `INVAR` et `TRANS`. Rappelons que ces dépendances peuvent concerner deux variables observables apparaissant dans l'invariant ou le lien entre une variable directement observable et une variable non-observable nécessaire à la formulation de l'invariant. Ainsi, l'automate $\mathcal{A}$ de la section 3.2.1.2 est défini.

Les interventions sont définies par instanciation du module `Intervention`.

```
Intervention(expr stat_precond, next expr seq_precond, expr condition,
    next expr effect)
```

Une intervention a trois paramètres définis par l'utilisateur. La précondition statique est une expression qui ne comporte aucun opérateur `next()` puisqu'elle porte sur les états d'application. L'effet est une contrainte sur l'état suivant l'état d'application donc il contient obligatoirement un opérateur `next()`. Il est exprimé depuis l'état d'application. Comme NuSMV n'a pas d'opérateur «précédent», l'expression de la pré-condition séquentielle est décalée d'un pas en avant. Ainsi, la pré-condition $prec\_v^\star = [0, V_0 - V_m[\wedge v^\star = [V_0 - V_m, V_0[$ est encodée par `v=0 & next(v)=1`. La précondition séquentielle porte sur la transition précédant l'état d'application, elle peut avoir un opérateur `next()` ou non. Les variables complémentaires d'historique (application de l'intervention et décalage permettant l'interprétation de la pré-condition séquentielle) sont gérées en interne par ce module. Nous obtenons alors l'automate $\mathcal{A}'$ défini à la section 3.2.2.

Le paramètre `condition` du module `Intervention()` sert d'interface pour définir la stratégie. Pour permettre la synthèse, ce paramètre sera systématiquement de la forme `flag_nameMyInterv`, où `nameMyInterv` est le nom de l'instance du module. La stratégie vide est par exemple définie par :

```
DEFINE flag_lock_arm := FALSE;
DEFINE flag_brake := FALSE;
```

Une fois la stratégie ajoutée au modèle, l'automate construit par NuSMV correspond à $\mathcal{A}_N$ défini à la section 3.2.3.

3.3.2.2 Partie générée et outil

L'utilisateur est en grande partie libéré de la tâche d'encodage des propriétés de sécurité, de permissivité et de validité. Une fois la variable `cata` définie, la propriété de sécurité est prédéfinie par `INVARSPEC ! cata`

Par ailleurs, nous avons développé un outil qui génère la validité et la permissivité par défaut à partir des noms des interventions et des noms et valeurs maximales des variables d'état.

La validité est générée via les paramètres des modules `Intervention()`. Les éventuels conflits entre des interventions modélisées au sein du même modèle doivent être ajoutés manuellement. Par exemple, pour un modèle avec trois interventions `brake`, `accelerate`, `klaxon`, la validité générée est enrichie par l'utilisateur d'un dernier terme déclarant le conflit entre le freinage et l'accélération.

```
INVARSPEC (flag_brake -> brake.stat_precond) & (flag_accelerate ->
    accelerate.stat_precond) & (flag_klaxon -> klaxon.stat_precond)
                -- default generated validity
    & !(flag_brake & flag_accelerate)   -- conflict
```

La permissivité générée est la permissivité par défaut, c'est-à-dire l'atteignabilité universelle de tous les états non-catastrophiques de $\mathcal{A}$. Rappelons que son espace d'état S est par construction (faiblement) connexe. Générer les propriétés de permissivité requiert donc de déterminer les états de $\mathcal{A}$ non-catastrophiques et atteignables, ce qui est fait par énumération des prédicats caractéristiques. L'outil fait appel à NuSMV pour vérifier pour chaque état $s \in \mathcal{P}_1 \times ... \times \mathcal{P}_n$ la propriété :

$$\Big(EF\ Carac(s)\Big)\ \wedge\ AG\Big(Carac(s) \Rightarrow \neg cata\Big)$$

Pour chaque état vérifiant cette propriété, c'est-à-dire pour chaque état atteignable et non-catastrophique, l'outil génère un appel au module `LiveProp`.

```
LiveProp(expr state, expr cata, expr diag)
```

Il contient les propriétés d'atteignabilité simple, universelle et continue telles que définies à la section 3.2.4. Le premier argument est le prédicat caractéristique de l'état considéré. Le deuxième argument est toujours valué à `cata`, et le troisième à `diag`, équivalent de la variable *diag* définie à la section 3.2.4 pour identifier les transitions diagonales. Les transitions diagonales sont gérées de manière prédéfinie en considérant que les variables sont indépendantes. En cas de dépendance, une transition correspondant à des changements simultanés de plusieurs variables dépendantes peut être déclaré comme non-diagonale par l'utilisateur.

En complément de la génération des appels à `LiveProp()`, un script est généré. Il lance la vérification de toutes les atteignabilités universelles des états non-catastrophiques, rendant la vérification de la permissivité atomique du point de vue de la synthèse. Après sa génération et avant la synthèse, ce script peut être modifié par l'utilisateur afin de personnaliser la permissivité. Comme nous le verrons au chapitre 4, la permissivité choisie doit au moins comprendre l'atteignabilité simple de chaque état non-catastrophique atteignable pour garantir des bonnes propriétés à l'algorithme de synthèse.

Notre outil de génération a également pour fonction de déterminer les états d'alerte automatiquement. C'est une fonctionnalité extrêmement utile puisqu'elle permet de donner à l'utilisateur un aperçu des chemins par lesquels le système

atteint un état dangereux. Comme une stratégie ne prend en compte que les états d'alerte, c'est aussi une étape préliminaire à la synthèse. Un état s de $\mathcal{A}$ non-catastrophique et atteignable est un état d'alerte si la propriété suivante est vraie :

$$EF\Big(Carac(s) \wedge EX\ cata\Big)$$

Pour faciliter la représentation de la stratégie en tant qu'association des interventions aux états d'alerte, le modèle est automatiquement complété avec des variables auxiliaires représentant le prédicat caractéristique de chaque état d'alerte. Dans le cas de l'exemple de la section 3.1, les états d'alerte sont :

```
DEFINE flag_st1 := r=0 & v=1;
DEFINE flag_st2 := r=1 & v=1;
DEFINE flag_st3 := r=1 & v=2;
```

L'étape de génération facilite l'écriture du modèle, notamment pour la permissivité, et prépare la synthèse en déterminant les états d'alerte et en permettant la vérification atomique de la permissivité.

3.4 Existence et choix d'une stratégie

Une stratégie est une association entre des états d'alerte et des interventions. Dans la section 3.2.2, on l'a formalisée par une fonction $N : S_A \rightarrow \mathcal{P}(I)$. Elle est représentée de manière équivalente au sein du modèle en NuSMV par l'affectation des conditions d'interventions à des ensembles d'états d'alerte. Par exemple, la stratégie présentée par la figure 3.3 est encodée par :

```
DEFINE flag_lock_arm := flag_st2 | flag_st3;
DEFINE flag_brake := flag_st1;
```

Le modèle, sans stratégie, est construit par l'utilisateur et l'outil de génération pour servir d'entrée à la synthèse, qui retourne plusieurs, voire toutes les stratégies correctes vis-à-vis des propriétés de sécurité, permissivité et validité. Le choix de la stratégie à adopter parmi les stratégies synthétisées permet de prendre en compte des informations qui n'apparaissent pas dans le modèle en raison de sa forme simple. Par exemple, des dépendances ou le coût d'implantation d'une intervention ou d'une observation du moniteur. Si la synthèse est complète, c'est-à-dire qu'elle retourne toutes les solutions, le fait qu'aucune stratégie correcte ne soit synthétisée permet de conclure à l'inexistence de stratégie correcte et donc à la nécessité de modifier le modèle. Déterminer la façon dont le modèle doit être modifié est complexe. L'utilisateur peut s'aider du simulateur de NuSMV pour explorer manuellement des traces dans le modèle et avoir une intuition du point de blocage (voir [Machin 2014]). Plusieurs options s'offrent à l'utilisateur pour modifier le système :

— Modifier le comportement, en particulier en ajoutant des dépendances. Des dépendances existantes peuvent avoir été omises lors de la première modélisation. Par ailleurs, faire des hypothèses d'environnement, modélisées

par des dépendances, aide potentiellement à trouver des stratégies en restreignant les chemins catastrophiques. Ces hypothèses devront ensuite être mises en œuvre concrètement dans le système.

— Ajouter des interventions. Lorsque le modèle n'est pas assez contrôlable, la sécurité ne peut pas être assurée. Il faudra donc ajouter des interventions, dans le modèle et le système.

— Réduire les exigences de permissivité. La sécurité ne peut parfois être assurée qu'au prix d'une perte de permissivité (en degré ou en état). Le mode verbeux de l'outil de synthèse peut aider à identifier ces cas.

— Modifier l'invariant de sécurité. Lorsque tous les recours ont été épuisés, on peut reconsidérer la définition des états catastrophiques, avec l'avis de l'équipe responsable de l'analyse de risque.

Plusieurs voies sont souvent possibles, un compromis doit être trouvé, par exemple entre le coût à la conception de la mise en place d'une intervention et le coût en opération d'une perte de permissivité.

3.5 Conflits entre stratégies

Un modèle, tel qu'il a été défini et encodé dans une instance du patron, représente *un* invariant de sécurité et tout ce qui permet de l'observer et de le couvrir. La stratégie est choisie pour satisfaire toutes les propriétés désirables. En particulier, au sein de la stratégie, les interventions sont appliquées de manière non-conflictuelle. Si les modèles évoluent en parallèle, les interventions sont déclenchées par le moniteur dans le système réel, unique. Des conflits peuvent donc apparaître entre des interventions appliquées par différentes stratégies. Repérer ces conflits implique de rassembler toutes les stratégies. Nous avons choisi de rassembler tous les modèles locaux (stratégies incluses), dans un modèle global afin d'évaluer les stratégies par l'effet qu'elles ont sur le système. Les variables observables comme les interventions sont peu nombreuses et donc très souvent utilisées dans plusieurs stratégies. Elles doivent avoir une évolution cohérente entre les différentes modèles locaux.

Techniquement, on rassemble dans le modèle global les modules principaux de toutes les instances du patrons, en y joignant les modules prédéfinis. Le modèle global a bien évidemment un module principal (qu'on appellera *module global* dans la suite). Il assure la cohérence des observations et des interventions et définit les propriétés globales à vérifier.

3.5.1 Mise en cohérence des observations

Pour chaque variable concrète v_j apparaissant dans *au moins un* invariant, les atomes de tous les invariants et de toutes les marges faisant apparaître v_j sont réunis. La procédure de partitionnement (présentée par la figure 3.6) est alors appliquée pour obtenir une partition globale $\mathcal{P}_j^g$ du domaine de définition D_j. La

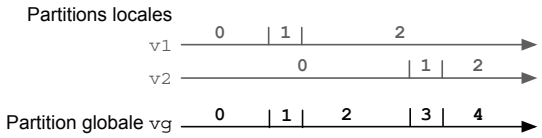


FIGURE 3.9 – Mise en cohérence d'une variable observable utilisée dans plusieurs instances

partition obtenue est illustrée par la figure 3.9 sur les variables encodées. (Comme les partitions locales, la partition globale est encodée par des entiers naturels.) À une valeur concrète dans D_j correspond une partie de chaque partition locale $\mathcal{P}_j$ et une partie de $\mathcal{P}_j^g$.

On définit une variable globale v_j^g abstraite qui sert à lier les variables locales et prend ses valeurs dans la partition globale $\mathcal{P}_j^g$. Par construction, les valeurs des variables locales $v_j^\star$ déterminent la valeur de v_j^g. Ceci est encodé par des contraintes entre variables globales et locales. Dans le cas de la figure 3.9, ces contraintes sont :

```
INVAR vg=0 <-> v1=0 & v2=0;
INVAR vg=1 <-> v1=1 & v2=0;
INVAR vg=2 <-> v1=2 & v2=0;
INVAR vg=3 <-> v1=2 & v2=1;
INVAR vg=4 <-> v1=2 & v2=2;
```

La définition des variables globales assure à la mise en cohérence des observations. Sur l'exemple de la figure 3.9, plaçons-nous à `v2=0 & v1=0`, on a alors `vg=0`. Par la continuité, la seule évolution possible de `v2` est `v2=1`, ce qui implique `vg=3` et `v1=2`. Cette évolution viole la continuité de `v1` et n'est donc pas possible. Au contraire, la seule évolution continue de `v1` est `v1=1`, ce qui implique `vg=1` et donc `v2=0`, ce qui est possible.

Les dépendances entre plusieurs observations sont modélisées de la même façon qu'au sein d'un modèle. Et, comme au sein d'un modèle local, la précision de la permissivité vérifiée dépend de l'effort investi dans la modélisation des dépendances.

3.5.2 Mise en cohérence des interventions

Comme elles sont peu nombreuses, les interventions sont souvent utilisées pour couvrir plusieurs invariants. Une même intervention est donc modélisée au sein de plusieurs modèles, vis-à-vis de plusieurs invariants. Il n'est pas possible de modéliser globalement une intervention car son effet et ses pré-conditions sont différents selon les variables observables (et leur partitions) définies dans le modèle local.

Lorsqu'une intervention est appliquée par une stratégie au sein d'un modèle local, cette intervention peut avoir des effets dans les autres modèles. L'application d'une intervention dépend de paramètres locaux comme les pré-conditions. La mise en cohérence du déclenchement se fait donc au moyen de l'association entre états d'alerte et interventions. Les modèles locaux sont modifiés afin que la

composante liée à la stratégie (locale) dans la condition d'application soit enrichie et tienne compte des autres stratégies. Il suffit que l'état dans lequel se trouve un automate local soit associé à une intervention i pour que, dans tous les autres automates locaux, la composante de la condition d'application de i liée à la stratégie deviennent vraie. Ceci ne préjuge pas de l'application effective de l'intervention dans les automates locaux, les pré-conditions pouvant ne pas être satisfaites.

Techniquement, le module global définit une variable auxiliaire d'association globale pour chaque intervention, qui correspond à l'union de toutes les variables d'association des modules locaux. La variable d'association globale de chaque intervention est passée en paramètre aux modules locaux. Elle est simplement ajoutée à la variable d'association locale dans la condition de l'intervention concernée. Par exemple, considérons deux modules `SI1` et `SI2` qui utilisent tous deux la même intervention.

```
MODULE SI1
myinterv1 : Intervention(flag_myinterv1);
MODULE SI2
myinterv2 : Intervention(flag_myinterv2);
```

L'intervention est modélisée par deux instances différentes `myinterv1` et `myinterv2` (dont l'interface est ici réduite au paramètre `condition`). La mise en cohérence se traduit par l'ajout d'une variable globale `g_myinterv`.

```
MODULE SI1(g_myinterv)
myinterv1 : Intervention(flag_myinterv1 | g_myinterv);
MODULE SI2(g_myinterv)
myinterv2 : Intervention(flag_myinterv2 | g_myinterv);
MODULE main
si1 : SI1(g_myinterv);
si2 : SI2(g_myinterv);
DEFINE g_myinterv := si1.flag_myinterv1 | si2.flag_myinterv2;
```

La mise en cohérence se fait bien au niveau de l'association et pas de l'application. L'effet et les pré-conditions restent en effet inchangés par la mise en cohérence et locaux aux instances du patron.

3.5.3 Vérifications

Les automates locaux évoluent maintenant de manière cohérente. Nous pouvons donc vérifier les propriétés du modèle global.

La première vérification à faire correspond à la sécurité au niveau global, c'est-à-dire à l'absence d'interactions conflictuelles entre stratégies. En notant g_i la variable d'association globale de l'intervention i et en considérant un ensemble d'interventions conflictuelles I_{confl}, il faut vérifier la propriétés de non-concomitance

$$AG\neg \bigwedge_{i \in I_{confl}} g_i$$

Par exemple, si le freinage et l'accélération sont utilisés dans des modèles locaux distincts, il faut vérifier `INVARSPEC !(g_brake & g_acceleration)`.

Dans le cas où un état de conflit existe, l'utilisateur le résout en ajoutant une priorité sur l'une ou l'autre des interventions. Dans ce cas, les invariants dans lesquels sont utilisées ces interventions doivent être modifiés pour en tenir compte. La permissivité globale est vérifiée par les propriétés d'atteignabilité locales.

3.6 Conclusion

Chaque invariant est modélisé par un automate auquel sont adjointes les interventions et les propriétés de sécurité, de permissivité et de validité. Pour être complet, le modèle doit également contenir une stratégie, fût-elle vide. Une stratégie associe des interventions à des états d'alerte, l'application effective des interventions dépendant aussi de leurs pré-conditions. Les interventions appliquées réduisent le comportement possible du système en coupant ou conditionnant des transitions. L'analyse de conflit entre stratégies est conduite sur la réunion des modèles de tous les invariants.

Rappelons que les interventions se divisent en deux classes : les inhibitions qui empêchent un changement d'état et les actions qui provoquent un changement d'état. Dans notre modélisation, forcer un changement d'état correspond à la suppression de la transition bouclant sur l'état. L'effet de l'intervention est alors supposé immédiat et correspond donc à une hypothèse forte. La plupart des actions sont donc modélisées comme des inhibitions avec des pré-conditions séquentielles.

Nous avons implémenté notre modélisation avec le vérificateur de modèle NuSMV. Un patron de modélisation a été développé en NuSMV, qui facilite la modélisation et permet une certaine normalisation du modèle encodé, exploitée par l'outil de génération que nous avons développé. La permissivité par défaut est générée automatiquement. Au besoin, l'utilisateur peut l'ajuster grâce à sa modélisation modulaire.

L'outil de génération est garant de certaines hypothèses que nous avons faites lors de la modélisation formelle. Les propriétés de permissivité ne sont générées que pour les états non-catastrophiques atteignables et les états d'alerte sont ceux qu'une seule transition sépare d'un état catastrophique. En déterminant automatiquement les états d'alerte, l'outil fait le lien entre le modèle issu du patron et les abstractions nécessaires à la synthèse. Des variables observables, il génère les états d'alerte, et des propriétés d'atteignabilité, il fait une commande atomique de vérification de la permissivité qui sera utilisée lors de la synthèse.

Ce qu'il faut retenir :

- Un modèle représente le comportement du système du point de vue du moniteur, vis-à-vis d'un invariant.
- La construction du modèle est facilitée par notre patron de modélisation.

- Les états d'alerte sont déterminés automatiquement.
- Les interventions suppriment des transitions.
- La complétude de la synthèse permet de conclure à l'inexistence de solutions et donc à la nécessité de modifier le modèle.
- L'analyse de conflit se fait sur la réunion des modèles définies pour les invariants.

Chapitre 4

Synthèse des stratégies de sécurité

Sommaire

Ce chapitre vise à présenter la façon dont les stratégies de sécurité sont synthétisées. Nous considérons qu'un modèle a été défini par l'utilisateur et que le comportement, les interventions et les propriétés sont fixés. Une vision entrées/sorties de la synthèse est donnée par la figure 4.1. Les stratégies étant des associations entre états d'alerte et interventions, nous avons besoin pour les synthétiser de la liste des interventions, saisie par l'utilisateur, et de la liste des états d'alerte, générée. La connaissance qu'a l'algorithme de l'automate est limitée à ces deux listes. L'algorithme ne dispose donc pas de toute la structure état-transition mais peut interagir avec le modèle par NuSMV. Ainsi la synthèse est plus robuste vis-à-vis

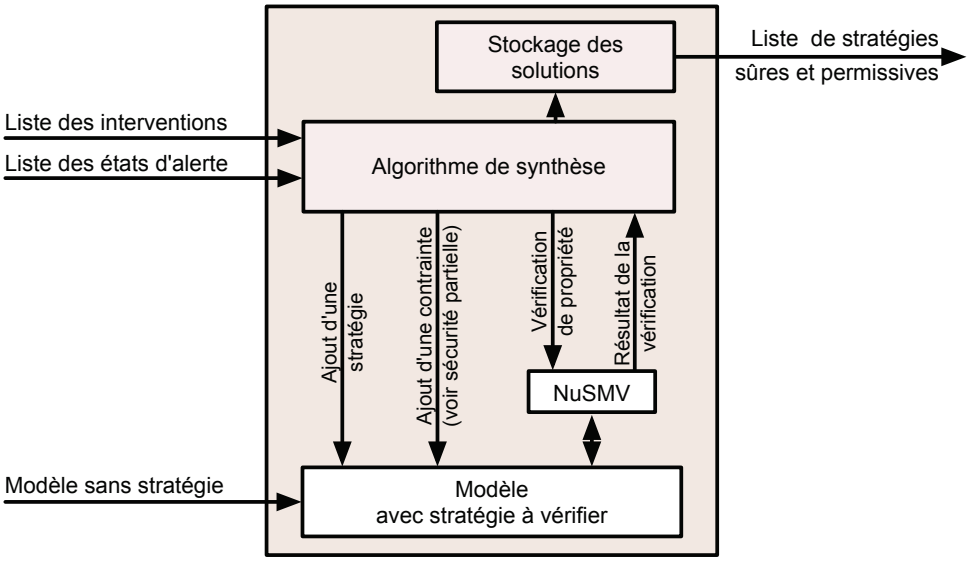


FIGURE 4.1 – Vision schématique de l'outil de de synthèse

des singularités du modèle ou de changements dans le patron, et la souplesse du langage de modélisation NuSMV peut être exploitée.

Au cours du processus de synthèse, les actions possibles vis-à-vis du modèle sont l'ajout d'une stratégie, la vérification des propriétés (par NuSMV) et, comme on le verra à la section 4.3.4, la définition d'un sous-modèle par l'ajout d'une contrainte. L'utilisateur ne peut pas modifier le modèle pendant la synthèse. Les états et degrés des exigences de permissivité étant fixes au cours de la synthèse, la permissivité est donc vue comme une propriété atomique et non comme la conjonction de plusieurs propriétés d'atteignabilité sur les différents états.

Le point de vue adopté pour la présentation de la synthèse est celui de l'ensemble de ses solutions, caractérisé par des propriétés comme la complétude. La synthèse est d'abord spécifiée dans la section 4.1. L'algorithme de synthèse est ensuite exposé dans son principe à la section 4.2. Il est constitué de critères de coupe dont la section 4.3 donne les propriétés en terme d'ensemble de solutions. Plusieurs variantes de l'algorithme ont été mises en œuvre dans l'outil de synthèse que nous avons développé. Nous en présentons les définitions et les performances dans la section 4.4. Enfin, la section 4.5 propose une approche alternative pour notre problématique, qui s'appuie sur la théorie des jeux.

4.1 Spécification de la synthèse

Avant de spécifier l'algorithme de synthèse par les stratégies solutions attendues, rappelons la formalisation des stratégies.

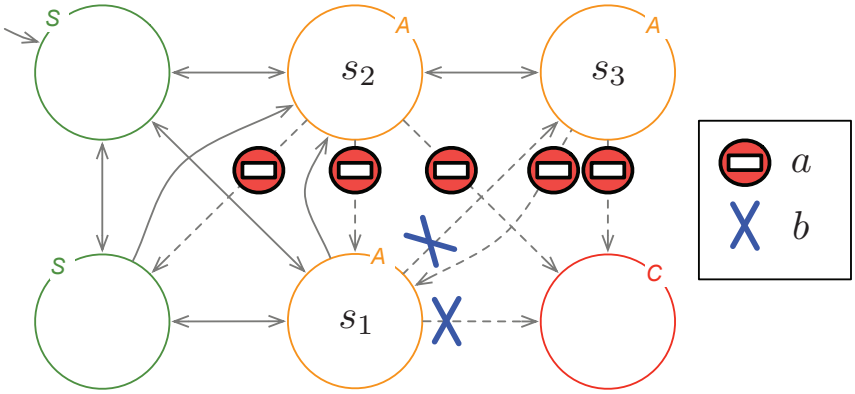


FIGURE 4.2 – Exemple pour la notation des stratégie

4.1.1 Formalisation des stratégies

Une stratégie N est formalisée comme une fonction

$$N : S_A \rightarrow \mathscr{P}(I)$$

avec S_A l'ensemble des n états d'alerte, I l'ensemble des m interventions, et $\mathscr{P}(I)$ l'ensemble des parties de I.

Sur l'exemple de la figure 4.2, on a $S_A = \{s_1, s_2, s_3\}$, $I = \{a, b\}$, et $\mathscr{P}(I) = \{\emptyset, \{a\}, \{b\}, \{ab\}\}$, qu'on abrègera en $\mathscr{P}(I) = \{\emptyset, a, b, ab\}$. La stratégie représentée est la fonction N telle que $N(s_1) = b$, $N(s_2) = a$ et $N(s_3) = a$, qu'on note $N = \{(s_1, b), (s_2, a), (s_3, a)\}$.

En tant qu'ensemble de parties, $\mathscr{P}(I)$ est doté d'une inclusion sur laquelle on s'appuie pour définir une inclusion au sein de l'ensemble des stratégies. Soient deux stratégies N, N' :

$$N \subset N' \iff \forall s \in S_A, N(s) \subseteq N'(s) \text{ et } \exists s \in S_A, N(s) \subset N'(s)$$

Les stratégies sont définies sur l'ensemble des états d'alerte S_A, c'est-à-dire un sous-ensemble de l'espace d'état S de l'automate $\mathcal{A}$. En revanche, les conditions d'application, et donc les effets des interventions, sont définis dans $\mathcal{A}_N$, d'espace d'état S'. Cet espace d'état est obtenu par raffinement de S avec des variables complémentaires d'historique à un pas, nécessaires à l'interprétation des pré-conditions séquentielles. La vérification des propriétés doit donc être effectuée au niveau de l'automate $\mathcal{A}_N$, ce qui est fait lorsque NuSMV vérifie le modèle encodé. Rappelons que les prédicats caractéristiques permettent d'établir une correspondance entre un état de S et les états de S' issus de son raffinement.

Un chemin entre deux états $s_1, s_2 \in S$, existant dans $\mathcal{A}$, est dit supprimé par la stratégie N si N coupe suffisamment de transitions de $\mathcal{A}_N$ pour qu'il n'existe plus de chemin entre aucune des versions raffinées des états s_1, s_2. En d'autres termes,

pour une stratégie N, il existe un chemin entre deux états s_1, s_2 de S, si il existe au moins un chemin dans $\mathcal{A}_\mathcal{N}$ entre des états s'_1 et s'_2 de S' tels que $Carac(s'_1) = Carac(s_1)$ et $Carac(s'_2) = Carac(s_2)$.

4.1.2 Ensembles de solutions

L'ensemble de solutions intuitivement attendu de la synthèse est l'ensemble $\mathcal{N}_{cor}$ des stratégies *correctes*, c'est-à-dire des stratégies sûres, permissives et valides (ces trois propriétés étant définies à la section 3.2.4) :

$$\mathcal{N}_{cor} = \{N : S_A \to \mathcal{P}(I) \mid N \text{ est } sur \wedge perm \wedge valid\}$$

Bien que correctes, certaines stratégies de $\mathcal{N}_{cor}$, dites *non-minimales*, ne sont pas vraiment pertinentes car elles comportent des interventions superflues. Ces interventions sont associées de telle façon qu'elles n'ont pas d'effet sur le comportement ou que leurs effets n'ont pas d'incidence sur les propriétés de sécurité et de permissivité. Il faut noter que, même si la permissivité telle qu'elle est modélisée n'est pas réduite, la permissivité au sens large l'est. Par exemple, un chemin entre deux états peut exister mais être plus long en présence d'interventions superflues. Par ailleurs, les stratégies non-minimales peuvent être très nombreuses, surchargeant le travail de choix d'une stratégie parmi les résultats de la synthèse.

On définit la minimalité d'une stratégie correcte et l'ensemble $\mathcal{N}_{min}$ des stratégies correctes et minimales par :

$$\text{Soit } N \in \mathcal{N}_{cor}, \quad N \in \mathcal{N}_{min} \iff \left(\forall N', N' \subset N \Rightarrow N' \notin \mathcal{N}_{cor}\right)$$

Remarquons que, s'il existe (au moins) une stratégie correcte, alors il existe (au moins) une stratégie minimale.

4.1.3 Propriétés de complétude et d'exclusivité

Nous avons défini deux ensembles de solutions différents. Cependant, il peut être difficile d'obtenir à la fois efficacement et exactement ces ensembles. Le compromis avec la performance amène à considérer des sur-approximations ou des sous-approximations de l'ensemble de solutions. Nous définissons donc deux propriétés caractérisant l'ensemble des solutions retourné par une synthèse vis-à-vis d'un ensemble de solution souhaité.

Une synthèse est dite *complète* vis-à-vis d'un ensemble de solutions ($\mathcal{N}_{cor}$ ou $\mathcal{N}_{min}$) lorsqu'elle retourne au moins l'ensemble des solutions considéré. En d'autres termes, elle retourne toutes les solutions, et éventuellement d'autres stratégies. Lorsqu'une synthèse est complète, l'absence de retour permet de conclure à la non-existence de solutions pour le modèle donné, et donc à la nécessité de modifier le modèle.

Une synthèse est dite *exclusive* vis-à-vis d'un ensemble de solutions ($\mathcal{N}_{cor}$ ou $\mathcal{N}_{min}$) lorsqu'elle retourne au plus l'ensemble des solutions considéré. Elle retourne

uniquement des stratégies qui sont solutions, mais peut en omettre. Une synthèse exclusive rend plus facile l'analyse des résultats et la sélection d'une stratégie en retournant moins de solutions et des solutions dont les propriétés sont garanties.

Remarquons que, comme $\mathcal{N}_{min} \subset \mathcal{N}_{cor}$, la complétude en $\mathcal{N}_{cor}$ implique la complétude en $\mathcal{N}_{min}$, et l'exclusivité en $\mathcal{N}_{min}$ implique l'exclusivité en $\mathcal{N}_{cor}$.

Notre objectif sera de démontrer les propriétés de complétude et d'exclusivité pour nos algorithmes de synthèse.

4.2 Algorithme de synthèse

L'approche de synthèse que nous adoptons est simple : énumérer et vérifier. L'algorithme le plus simple serait d'énumérer exhaustivement les stratégies possibles, c'est-à-dire les 2^{nm} associations possibles entre l'ensemble des n états d'alerte et l'ensemble des 2^m combinaisons d'interventions, et de vérifier chaque stratégie. Cet algorithme serait complet et exclusif en $\mathcal{N}_{cor}$. Il serait aussi extrêmement long à exécuter.

Pour accélérer l'exploration de l'ensemble des stratégies, nous choisissons de considérer des stratégies partiellement spécifiées, c'est-à-dire ne faisant pas de choix quant aux interventions à associer à certains états d'alerte. En effet, l'évaluation des propriétés des stratégies partielles permet d'éviter d'énumérer de nombreuses stratégies totalement spécifiées. Un arbre, dont les nœuds sont des stratégies totales ou partielles, structure l'ensemble des stratégies considérées.

4.2.1 Arbre de stratégie

Pour considérer les stratégies partielles, on étend la définition des stratégies par :

$$N : S_A \rightarrow \mathscr{P}(I) \cup \{\bot\}$$

avec $\bot$, l'indéfini dans l'association des interventions. L'indéfini signifie que le choix de la combinaison d'interventions n'est pas fait. Ce choix est reporté à l'exploration d'autres nœuds de l'arbre.

Une stratégie N est dite totale si $\forall s, N(s) \neq \bot$. Les stratégies totales jouent le rôle de feuille de l'arbre. Un nœud interne est une stratégie partielle, telle que $\exists s, N(s) = \bot$. On définit l'opérateur *tot* pour obtenir la version totale d'une stratégie partielle en remplaçant $\bot$ par $\emptyset$:

$$\forall s \in S_A,\ tot(N)(s) = \begin{cases} \emptyset & \text{si } N(s) = \bot \\ N(s) & \text{sinon} \end{cases}$$

Les propriétés (sécurité, permissivité, validité) d'une stratégie partielle N sont définies comme les propriétés de sa version totale $tot(N)$.

Pour étendre l'inclusion des stratégies totales à tous les nœuds de l'arbre, on étend l'inclusion de $\mathscr{P}(I)$ à $\mathscr{P}(I) \cup \{\bot\}$ en posant $\bot \subset \emptyset$.

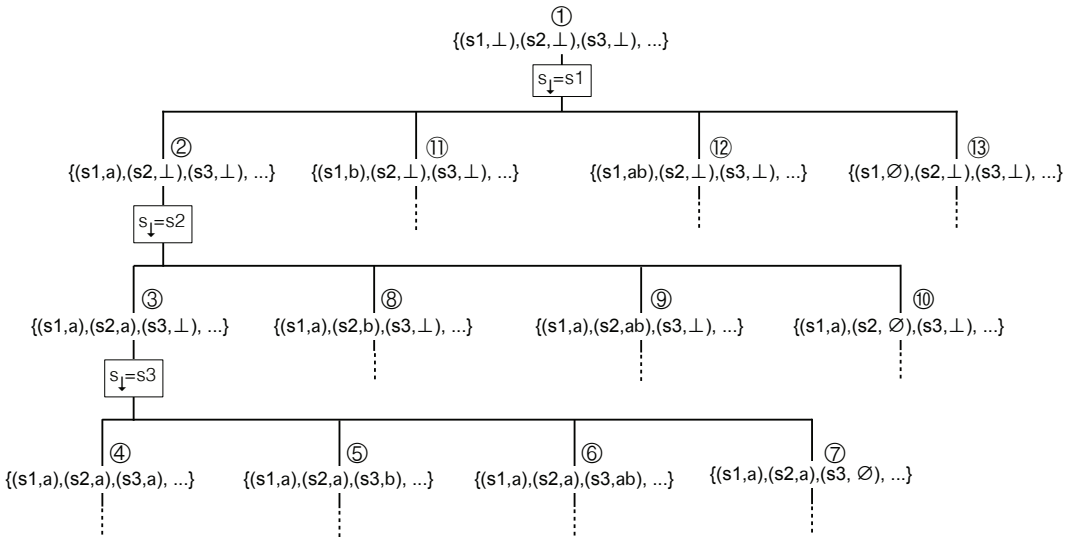


FIGURE 4.3 – Exemple de structure et de parcours d'arbre avec $I = \{a, b\}$. Les nœuds sont parcourus dans l'ordre de numérotation.

La construction de l'arbre des stratégies est illustrée par la figure 4.3. L'arbre a pour racine la stratégie indéfinie $N_0 = \{(s_1, \perp), ..., (s_n, \perp)\}$. Soit N une stratégie partielle de l'arbre. Pour construire les nœuds fils de N, on choisit arbitrairement un état parmi tous les $s \in S_A$ tels que $N(s) = \perp$. Cet état est dit *de descendance* et noté $s_\downarrow$. Les fils N_i de N par $s_\downarrow$ sont les 2^m stratégies définies par :

$$\forall s \in S_A,\ N_i(s) = \begin{cases} N(s) & \text{si } s \neq s_\downarrow \\ i & \text{si } s = s_\downarrow \text{ , avec } i \in \mathscr{P}(I) \end{cases}$$

Les nœuds d'un arbre forment un sous-ensemble de stratégies, l'arbre est donc fini. Les nœuds internes de l'arbre sont un sous-ensemble de stratégies partielles, qui dépend de l'état de descendance choisi à chaque nœud interne. En revanche, les feuilles de l'arbre constituent exactement l'ensemble des stratégies totales.

4.2.2 Relation entre les stratégies

Deux nœuds N', N'', fils de N par l'état de descendance $s_\downarrow$, sont dits frères. De plus, N'' est un *frère combiné* de N' si $N'(s_\downarrow) \subset N''(s_\downarrow)$.

Pour un nœud N, on appelle ses descendants, ses frères combinés et leurs descendants, les *nœuds postérieurs* à N. Les relations entre les nœuds impliquent des propriétés d'inclusion :

— N' descendant de $N \implies N \subset N'$. Par exemple, sur la figure 4.3, le nœud $N_4 = \{(s_1, a), (s_2, a), (s_3, a), ...\}$ est le descendant du nœud $N_2 = \{(s_1, a), (s_2, \perp), (s_3, \perp), ...\}$. On a bien $\perp \subset a$ pour les interventions associées à s_2 et s_3

— N' descendant de N et $N(s) \neq \perp \implies N'(s) = N(s)$. Pour l'état s_1, $N_2(s_1) \neq \perp$ et $N_4(s_1) = N_2(s_1) = a$.

— N' frère combiné de $N \implies N \subset N'$. Le nœud N_{12}, frère combiné de N_2, est inclut dans N_2 car l'intervention a est incluse dans ab.

Dans notre structure d'arbre, la profondeur est constante c'est-à-dire que toutes les feuilles ont le même nombre d'ancêtres. Le lien entre l'arbre et l'automate est explicite : les frères correspondent à différentes combinaisons d'interventions sur un même état d'alerte. Enfin, notre structure d'arbre permet d'appliquer des critères de coupe complémentaires sur les descendants et les frères combinés, comme ceux définis aux sections 4.3.4 et 4.3.5.

4.2.3 Parcours de l'arbre

On ne s'intéresse pas ici à l'ordre exact du parcours d'arbre mais seulement aux ensembles de solutions qu'il va retourner. Comme illustré par la figure 4.3, l'arbre est parcouru en profondeur préfixe. En particulier, un nœud n'est exploré que si tous ses ancêtres l'ont déjà été. De plus notre parcours est tel que les frères combinés d'une stratégie sont explorés après elle (sauf pour la stratégie telle que $N(s_\downarrow) = \emptyset$). Le parcours est fini, ce qui en garantit la terminaison.

Afin de rendre la synthèse performante, il faut explorer le moins de nœuds possibles ; on utilise donc des critères de coupe pour éliminer des branches. Pour un nœud examiné donné, les coupes portent sur son sous-arbre ou sur ses frères combinés et leurs descendants, ou sur l'ensemble des deux (les stratégies postérieures). Les critères de coupe sont vérifiés sur les nœuds internes et s'appuient fortement sur les propriétés déjà définies de sécurité, permissivité et de validité.

Comme l'arbre contient des stratégies partielles, qui ne peuvent pas être solutions, son exploration exhaustive serait plus longue que la simple énumération des stratégies totales, avec, dans les deux cas, $\mathcal{N}_{cor}$ retourné exactement. Cependant, la structure d'arbre permet d'utiliser des critères de coupe qui, vérifiés au niveau des nœuds internes, évitent d'explorer des sous-arbres et accélèrent donc la synthèse. En outre, cette structure est utile pour se rapprocher de la complétude et l'exclusivité en $\mathcal{N}_{min}$.

4.3 Critères de coupe

Les critères de coupe permettent de réduire la longueur du parcours d'arbre en prenant en compte la spécificité de notre problème. Les critères sont illustrés sur l'exemple de la figure 4.4 et accompagnés d'une explication des propriétés de complétude et d'exclusivité qu'ils garantissent à l'algorithme de synthèse.

4.3.1 Critère de permissivité

Le premier critère concerne la permissivité. La non-permissivité d'un nœud N peut être interprétée comme le fait que la stratégie supprime trop de transitions. Or, les nœuds postérieurs à N ont plus d'interventions, et donc moins de transitions.

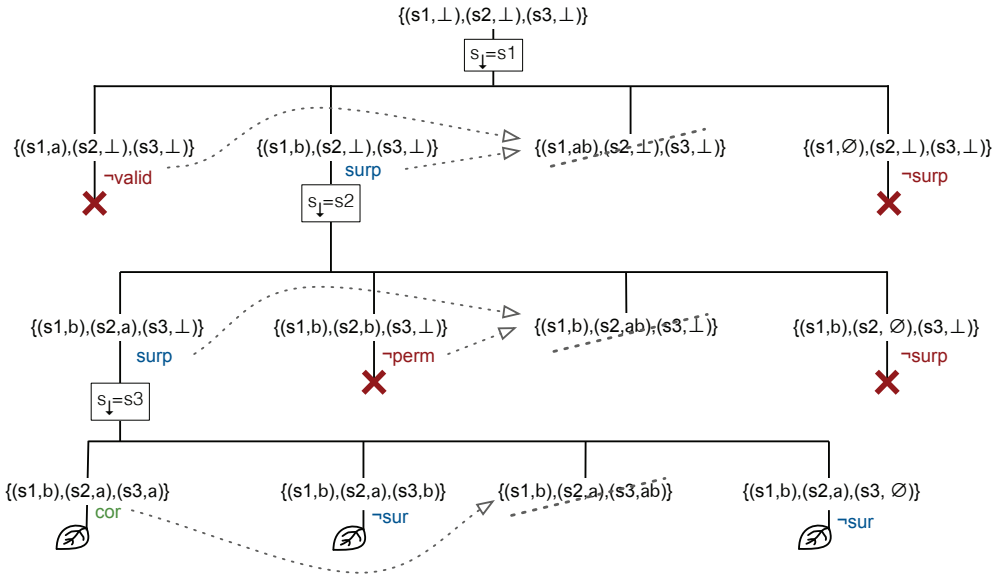


FIGURE 4.4 – Arbre des stratégies explorées pour la synthèse de l'exemple. Les définitions des états d'alerte et des interventions sont données par la figure 4.2.

Les nœuds postérieurs à une stratégie non-permissive sont donc non-permissifs. Les explorer n'aboutirait à aucune stratégie permissive, et n'est donc d'aucune utilité.

Dans la figure 4.4, la stratégie $\{(s_1, b), (s_2, b), (s_3, \bot)\}$ n'est pas permissive. En effet, comme le freinage b est appliqué quand la vitesse est dans la marge (états s_1 et s_2 de la figure 4.2), l'état s_3 n'est plus atteignable, c'est-à-dire que la plate-forme ne peut jamais se déplacer à une vitesse supérieure à V_0. Nous coupons à la fois le sous-arbre de cette stratégie et son frère combiné $\{(s_1, b), (s_2, ab), (s_3, \bot)\}$.

Critère $\neg perm$ Si un nœud est non-permissif, tous ses nœuds postérieurs sont coupés.

Ce critère ne coupe que des stratégies non-permissives, donc incorrectes. Il conserve la complétude en $\mathcal{N}_{cor}$.

4.3.2 Critère de validité

La validité, définie à la section 3.2.4, permet la définition d'un deuxième critère de coupe. Si un nœud N est non-valide, cela signifie qu'un état s atteignable est associé à une combinaison conflictuelle d'interventions ou à une intervention dont la pré-condition statique n'est pas satisfaite en s. Dans un nœud N' postérieur à N, l'association de s à cette intervention ou combinaison d'intervention demeure. Si s est toujours atteignable dans N', alors N' est invalide. Sinon, s n'est pas atteignable et N' est non-permissive sous réserve que les exigences de permissivité contiennent l'atteignabilité simple de s.

Dans la figure 4.4, considérons la stratégie $\{(s_1, a), (s_2, \perp), (s_3, \perp)\}$. Elle est non-valide car elle associe le blocage du bras a à l'état s_1, dans lequel le bras est déplié. Les descendants associent des interventions à d'autres états d'alerte en conservant le blocage a en s_1 ; ils sont donc invalides. Par ailleurs, le frère combiné $\{(s_1, ab), (s_2, \perp), (s_3, \perp)\}$ associe également a à s_1, il est donc invalide. En tant que descendants d'un nœud invalide, ses descendants le sont aussi. Les nœuds postérieurs à une stratégie invalide sont invalides.

Critère $\neg valid$ Si un nœud est invalide, ses nœuds postérieurs sont coupés.

Ce critère ne coupe que des stratégies incorrectes. Si les exigences de permissivité contiennent au moins l'atteignabilité simple de tous les états non-catastrophiques, il conserve la complétude de la synthèse en $\mathcal{N}_{cor}$.

4.3.3 Critère de correction

Les critères précédents garantissent des propriétés vis-à-vis de $\mathcal{N}_{cor}$, mais on préfèrerait restreindre les solutions à $\mathcal{N}_{min}$. Si un nœud N est correct, la seule solution minimale postérieure à N est $tot(N)$. En effet, les nœuds postérieurs à N associent tous plus d'interventions que $tot(N)$. Dans le cas où un nœud postérieur à N serait correct, il ne serait donc pas minimal.

Le fait que N soit correcte est en fait déterminé par la vérification des propriétés de $tot(N)$. Les solutions de la synthèse ne pouvant être que des stratégies totales, on retient $tot(N)$ comme solution.

Critère *cor* Si N est correct, ses nœuds postérieurs sont coupés (au sens où ils ne sont ni explorés, ni vérifiés). Le nœud $tot(N)$ est stocké comme solution.

Ce critère coupe des stratégies incorrectes et des stratégies correctes non-minimales. Il rompt la complétude en $\mathcal{N}_{cor}$ mais conserve la complétude en $\mathcal{N}_{min}$. Il n'est pas suffisant pour l'exclusivité en $\mathcal{N}_{min}$.

4.3.4 Critère de sécurité partielle et coupe du sous-arbre

Pour les stratégies incorrectes, nous n'avons pas exploité la propriété de sécurité. Une stratégie non-sûre ne permet pas de couper des stratégies, car déployer des stratégies postérieures, c'est-à-dire ajouter des interventions, améliore *a priori* la sécurité. La sécurité globale ne permet donc pas de définir un critère de coupe. Avoir un critère partiel de sécurité permettrait d'anticiper le fait qu'une intervention soit efficace ou non, et donc de couper des sous-arbres.

On sait que les états d'alerte qui ne sont pas encore explorés (c'est-à-dire tels que $N(s) = \perp$) n'ont pas encore d'intervention. Donc, en passant par eux, le système peut atteindre un état catastrophique. L'idée est d'évaluer la sécurité du modèle sans ces états d'alerte qui mènent, *a priori*, à un état catastrophique en un pas. Cette version locale, ou préliminaire, de la sécurité est nommée *sécurité partielle*, et est notée *surp*. Elle permettra de définir des critères de coupe. La sécurité partielle

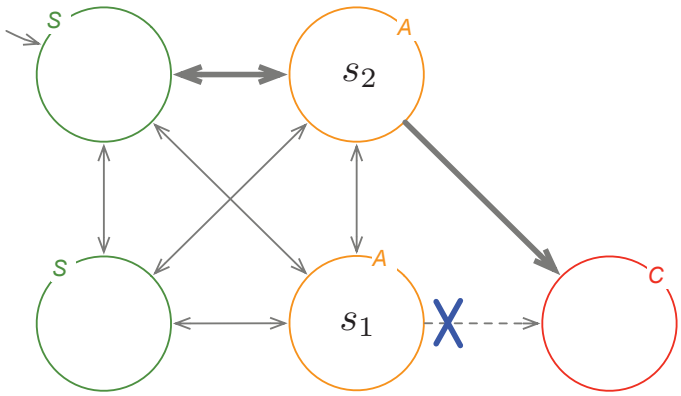


FIGURE 4.5 – Sous-modèle utilisé pour la vérification de la sécurité partielle de la stratégie $N = \{(s_1, b), (s_2, \emptyset), (s3, \bot)\}$. La stratégie N n'est pas partiellement sûre puisqu'il existe un chemin vers l'état catastrophique C (en gras).

consiste à vérifier la propriété de sécurité sur un modèle dont on a supprimé les états d'alerte non encore déployés dans l'arbre. Par exemple, la figure 4.5 illustre le sous-modèle [1] utilisé pour la vérification de la stratégie $N = \{(s_1, b), (s_2, \emptyset), (s3, \bot)\}$.

La sécurité partielle est donc la sécurité d'un sous-modèle. Comme il manque des états et des transitions au sous-modèle, la sécurité du modèle entier implique la sécurité du sous-modèle. La sécurité partielle est donc bien une version faible de la sécurité.

Remarquons que la sécurité partielle d'une stratégie N et de sa version totale $tot(N)$ peuvent être différentes. La non-intervention $\emptyset$ marque le fait qu'il a été décidé de ne pas appliquer d'intervention dans un certain état d'alerte, ce qui peut faire partie d'une stratégie totale. Contrairement aux états associés à $\bot$, ceux associés à $\emptyset$ ne sont pas retranchés du modèle lors de la vérification de la sécurité partielle. Par exemple, vérifier la sécurité partielle de $N = \{(s_1, a), (s_2, \emptyset), (s_3, \bot)\}$ se fait sur le sous-modèle ne contenant pas s_3, alors que la sécurité partielle de $tot(N) = \{(s_1, a), (s_2, \emptyset), (s_3, \emptyset)\}$ se ferait sur le modèle complet.

Soit une stratégie N qui n'est pas partiellement sûre. Dans le sous-modèle utilisé pour la vérification de la sécurité partielle de N, il existe un chemin π qui passe par un état d'alerte s puis par un état catastrophique. Soit N', un descendant N. Le chemin π est conservé car N' ne coupe que des transitions sortant d'états qui ne sont pas dans le sous-modèle. Le chemin π conduisant dans un état catastrophique, N' n'est pas sûr.

Critère $\neg surp$ Si N n'est pas partiellement sûr , ses descendants sont coupés.

Ce critère coupe que des stratégies non-sûres. Il conserve donc la complétude de la synthèse en $\mathcal{N}_{cor}$.

1. Dans le modèle NuSMV, la suppression des états d'alerte est effectuée par l'ajout automatique d'une contrainte. La suppression de s_3 s'écrit par exemple `INVAR !(flag_st3)`.

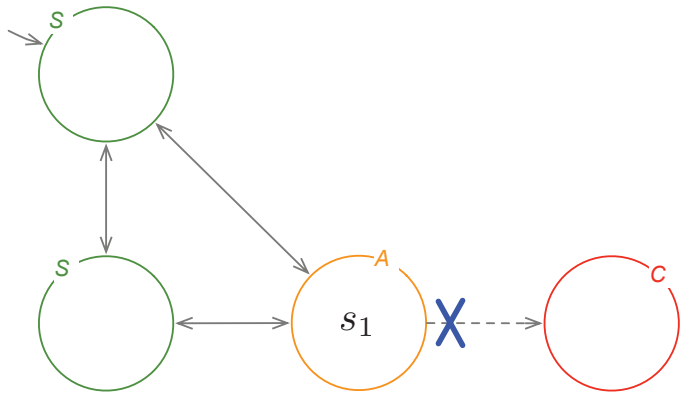


FIGURE 4.6 – Sous-modèle utilisé pour la vérification de sa sécurité partielle de $N_p = \{(s_1, b), (s_2, \perp), (s3, \perp)\}$. N_p est sous-sûre car il n'existe aucun chemin vers C dans le sous-modèle.

4.3.5 Critère de sécurité partielle et coupe des frères combinés

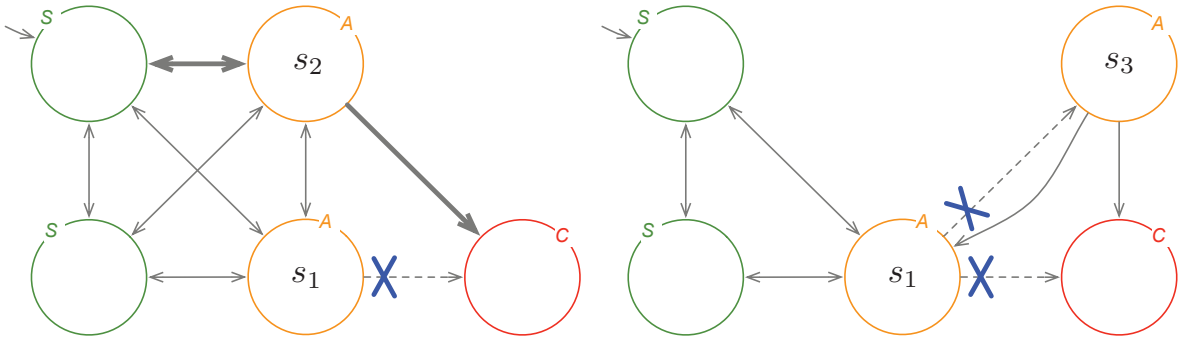
On souhaite réduire l'ensemble de solutions retourné, idéalement à $\mathcal{N}_{min}$. Pour ce faire, on envisage le critère coupant les frères combinés des stratégies partiellement sûres.

Illustrons le principe de ce critère sur la stratégie $N_p = \{(s_1, b), (s_2, \perp), (s3, \perp)\}$ de la figure 4.6. La stratégie N_p est sous-sûre car il n'existe aucun chemin vers l'état catastrophique C dans le sous-modèle. Comme C n'est plus directement atteignable depuis s_1, ajouter n'importe quelle intervention en s_1 paraît inutile : cela mènerait à des stratégies qui, si elles étaient correctes, ne seraient pas minimales. En d'autres termes, les frères combinés de N_p peuvent être coupés sans perdre de stratégies minimales.

Mais ce raisonnement présente une faille : il ne prend en compte que les chemins dans le sous-modèle. Dans le modèle complet, il peut exister des chemins qui justifient d'associer des interventions supplémentaires à s_1.

En premier lieu, considérons le cas simple dans lequel un état d'alerte n'est pas atteignable dans le sous-modèle, alors qu'il l'est dans le modèle complet. La stratégie N_p n'étant pas dans ce cas, considérons $N_f = \{(s_1, b), (s_2, \perp), (s3, c)\}$, le fils de N_p par l'état de descendance s_3. La figure 4.7b montre qu'il n'existe pas de chemin vers l'état catastrophique dans le sous-modèle de N_f et ce, quelque soit l'intervention c. La stratégie N_f est donc partiellement sûre. Mais de toute évidence, cette sécurité partielle ne dépend pas de l'intervention c ; elle est dite triviale et n'apporte pas d'information justifiant la coupe des frères combinés. Pour éviter la sécurité partielle triviale, il suffit de choisir comme état de descendance s_2, comme dans la figure 4.7a). Pour être partiellement sûrs, les fils de N_p par s_2 doivent associer à s_2 une intervention adéquate.

Afin d'éviter les cas de sécurité partielle triviale, nous ajoutons une contrainte sur le choix de l'état de descendance $s_\downarrow$: il doit être tel que le sous-modèle, comprenant $s_\downarrow$ mais sans intervention en $s_\downarrow$, ne soit pas sûr. Dans notre outil de synthèse, cette vérification est effectuée à chaque choix d'état de descendance. Remarquons



(a) Sous-modèle considéré si $s_{\downarrow} = s_2$ (b) Sous-modèle considéré si $s_{\downarrow} = s_3$

FIGURE 4.7 – Sous-modèles utilisés pour la vérification de la sécurité partielle des fils de $N_p = \{(s_1, b), (s_2, \perp), (s3, \perp)\}$ selon l'état de descendance choisi.

que cette contrainte sur le choix de l'état de descendance rend plus efficace le critère $\neg surp$ défini à la section précédente. En effet, comme la sécurité partielle n'est jamais trivialement vraie, le critère $\neg surp$ permet de couper plus souvent.

En second lieu, considérons des cas plus subtils faisant appel aux pré-conditions séquentielles. Après avoir coupé les frères combinés de N_p, ses fils sont construits par l'état de descendance s_2. Supposons que la seule intervention valide en s_2 a une pré-condition séquentielle : l'état précédent doit être différent de s_1. Ajouter des interventions à s_2 ne permet donc pas d'éliminer tous les chemins catastrophiques passant par s_2 ; il restera ceux en provenance de s_1. Aucune stratégie correcte ne sera donc trouvée parmi les stratégies postérieures à N_p. Pourtant, il peut en exister. Pour les trouver, il aurait fallu, lors du choix des interventions associées à s_1, considérer des interventions qui n'ont pas seulement une incidence sur la transition de s_1 à C mais aussi sur la transition de s_1 à s_2.

Ainsi, même en évitant les cas de sécurité partielle, couper les frères combinés des stratégies partiellement sûres peut amener à perdre des stratégies minimales. Le critère rompt donc la complétude en $\mathcal{N}_{min}$. Dans la plupart des cas, ce critère permet tout de même d'obtenir des solutions. Par exemple, associé à la non-trivialité de la sécurité partielle , il rend la synthèse complète en $\mathcal{N}_{min}$ dans le cas particulier de modèles sans aucune pré-condition séquentielle. En effet, le chemin parcouru ne conditionne alors plus l'effet de l'intervention. Pour une stratégie et un sous-modèle donné, couper des chemins du modèle complet, en plus des chemins du sous-modèle, n'influe pas sur les effets des interventions qui seront associées à des états n'appartenant pas au sous-modèle. Dans l'exemple précédent, si l'intervention associée à s_2 n'a pas de pré-condition séquentielle, associer à s_1 une intervention coupant la transition vers s_2 n'a pas d'intérêt. Les frères combinés d'une stratégie sous-sûre ne permettent donc pas de trouver des stratégies minimales en l'absence de pré-conditions séquentielles.

Qu'il y ait ou non des pré-conditions séquentielles, ce critère, combiné au critère *cor*, garantit que toutes les stratégies retournées sont minimales. Les nœuds non-

minimaux sont soit des descendants de nœuds minimaux, soit des frères combinés (ou descendants de frères combinés) de nœuds minimaux. Dans le premier cas, ils sont coupés par le critère *cor*, dans le deuxième, ils sont les frères combinés (ou descendants de frères combinés) d'un nœud partiellement sûr , et sont donc coupés par le présent critère.

Critère *surp* Si N est partiellement sûre , ses frères combinés et leurs descendants sont coupés.

Si les états de descendance sont choisis tels qu'il n'y ait pas de sécurité partielle triviale et si le critère *cor* est appliqué, alors ce critère rend la synthèse exclusive et non-complète en $\mathcal{N}_{min}$

4.4 L'outil de synthèse

Nous travaillons désormais sous les hypothèses que les exigences de permissivité comprennent au moins l'atteignabilité simple de tous les états non-catastrophiques et que les états de descendance sont choisis tels que la sécurité partielle n'est jamais trivialement vraie. A partir des critères définis ci-dessus, trois parcours d'arbre différents sont définis et réalisés par l'outil de synthèse que nous avons développé.

4.4.1 Les variantes du parcours d'arbre

Les trois variantes de la synthèse sont définies dans le tableau 4.1 ; leurs propriétés sont données dans le tableau 4.2.

La SC (Synthèse Complète) coupe selon les critères $\neg valid$, $\neg perm$, et $\neg surp$. Par la simple vérification des stratégies, la SC retourne exclusivement des stratégies correctes. Comme le garantissent les critères de coupe utilisés, la SC retourne toutes les stratégies correctes. Comme la SC est complète en $\mathcal{N}_{cor}$, elle est naturellement complète en $\mathcal{N}_{min}$.

La SCM (Synthèse Complète en solutions Minimales) utilise un critère de coupe supplémentaire par rapport à la SC. Le critère *cor* est ajouté. En rompant la complétude en $\mathcal{N}_{cor}$, il permet de se rapprocher de $\mathcal{N}_{min}$ dont il conserve la complétude.

La SEM (Synthèse Exclusive en solutions Minimales) ajoute le critère *surp*. Elle rend donc la synthèse exclusive vis-à-vis de $\mathcal{N}_{min}$. Cependant, pour un modèle comprenant des pré-conditions séquentielles, ce critère fait perdre la complétude en $\mathcal{N}_{min}$.

La SC a un intérêt théorique car elle retourne exactement $\mathcal{N}_{cor}$. Elle est longue et renvoie des stratégies non-minimales. On préfèrera donc en pratique la SCM et la SEM qui retournent respectivement un sur-ensemble et un sous-ensemble de $\mathcal{N}_{min}$. Le choix entre la SCM et la SEM est déterminé par plusieurs facteurs. Si l'espace de recherche est important, et que le comportement comporte peu de dépendances entre variables, les stratégies correctes seront *a priori* nombreuses.

Synthèse			Propriété du nœud examiné	Relation entre le nœud examiné et les nœuds coupés
SEM	SCM	SC	$\neg valid$	nœuds postérieurs
			$\neg perm$	nœuds postérieurs
			$\neg surp$	descendants
			cor	nœuds postérieurs
			$surp$	frères combinés et leurs descendants

Tableau 4.1 – Coupes réalisées dans les différentes synthèses

	$\mathcal{N}_{cor}$		$\mathcal{N}_{min}$	
	Exclusivité	Complétude	Exclusivité	Complétude
SC	√	√		√
SCM	√			√
SEM	√		√	

Tableau 4.2 – Propriétés des synthèses

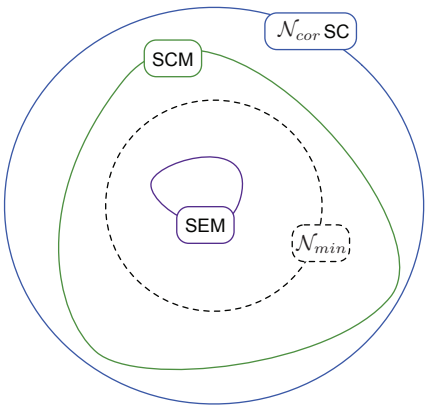


Figure 4.8 – Inclusions des ensembles de solutions et des ensembles retournés

Appliquer la SCM sera alors coûteux en temps de calcul et en temps d'analyse des résultats, pour sélectionner la stratégie à adopter. Dans ce cas, et plus généralement lorsqu'une stratégie minimale suffit, on préférera la SEM. La complétude de la SCM permet de statuer sur l'existence d'une stratégie correcte. Si la SCM ne retourne pas de solution, il n'y a pas de stratégie correcte possible dans le modèle. Il faut alors modifier le modèle (c.f. section 3.4).

4.4.2 Implémentation

Les trois variantes ont été implémentées dans un outil que nous développé en C/C++. Notre outil s'appuie sur NuSMV pour la vérification des critères de coupes et des propriétés. Il existe en deux versions, une en programmation purement séquentielle et une en programmation parallélisée, qui améliore les performances.

Dans la version parallélisée, plusieurs fils d'exécution sont définis, dont un, dit maître. Le maître manipule l'arbre et les nœuds en tant qu'associations. Les

Modèle	Nombre de stratégies	SCM % de nœuds explorés	SCM Nombre de solutions	SCM Temps d'exécution	SEM % de nœuds explorés	SEM Nombre de solutions	SEM Temps d'exécution
`2var_2A`	64	4,7	0	50ms	4,7	0	50ms
`2var_2I`	64	12,5	1	230ms	12,5	1	150ms
`2var_4`	4 096	0,56	9	210ms	0,44	6	170ms
`3var_3A`	10^6	10^{-3}	0	200ms	10^{-4}	0	200ms
`3var_3I`	10^6	10^{-2}	40	3,1s	10^{-3}	12	1,0s
`3var_6`	10^{12}	10^{-6}	17 106	4min	10^{-8}	1 128	18,3s
`4var_4A`	10^{18}	10^{-14}	0	3s	10^{-14}	0	2,7s
`4var_4I`	10^{18}	-	-	-	10^{-11}	12 954	26min

Tableau 4.3 – Performances de notre outil de synthèse

esclaves traduisent les nœuds en stratégies NuSMV et vérifient leurs propriétés.

Le maître génère une liste de nœuds, par exemple les enfants du nœud racine N_0 qui associent une intervention simple (non-combinée). Chaque esclave traite un nœud. Il dispose d'une copie locale du modèle, auquel il ajoute la stratégie et d'une interface avec NuSMV pour lancer la vérification. Les critères de coupes déterminent une politique de déploiement (couper ou déployer le sous-arbre, les frères combinés). Si le nœud est correct, l'esclave le stocke. La politique de déploiement, ainsi qu'un état de descendance évitant la sécurité partielle triviale, sont renvoyés par l'esclave au maître. Ces informations servent au maître à déployer les nœuds postérieurs au nœud traité, que les esclaves traiteront à leur tour.

4.4.3 Performances

Afin d'évaluer les performances de la version parallélisée de l'outil, nous avons réalisé des tests présentés dans le tableau 4.3. Seules les variantes SCM et SEM sont testées, la SC présentant peu d'intérêt pratique. Les tests de performance sont conduits sur des modèles artificiels, permettant de considérer des espaces de recherche de quelques dizaines de stratégies à plus de 10^{18}. Les modèles, classés par nombre de variables, sont générés comme suit. Une des variables a 3 valeurs, les autres en ont 2, ce qui constitue des nombres de valeurs courants dans les modèles réels. L'état initial est l'état où toutes les variables sont nulles, et l'état catastrophique, unique, est l'état où toutes les variables sont à leur valeur maximale. Trois possibilités sont explorées pour les interventions. Le suffixe `A`, pour «action», signifie qu'une action forçant la décrémentation est définie pour chaque variable. Le suffixe `I` est le symétrique pour les inhibitions : un blocage est défini pour chaque variable. Enfin, on considère dans un même modèle à la fois les actions et les inhibitions. Par exemple, `3var_6` a 3 variables et 6 interventions (3 actions et 3 inhibitions).

Les tests sont effectués sur un processeur Intel Core I7-4770 à 3.4GHz avec 16 GB de mémoire. Les résultats sont présentés dans le tableau 4.3. Pour chaque modèle, est mentionné le nombre de stratégies totales, c'est-à-dire de feuilles de l'arbre, ou encore d'étapes d'une recherche exhaustive. On veut évaluer le gain

de l'exploration de l'arbre avec des critères de coupe par rapport à la recherche exhaustive. La première colonne donne pour cela le pourcentage de nœuds explorés par rapport au nombre de stratégies totales. Comme nous explorons des nœuds internes, cette valeur pourrait être supérieure à 100%. Cependant, les valeurs sont très basses, ce qui démontre l'efficacité de nos critères de coupe. La deuxième colonne donne le nombre de solutions retournées. Les résultats de la variante SCM montrent qu'il n'y a pas de solutions dans les modèles qui ont seulement des actions comme interventions (suffixe `A`). Intuitivement, comme les actions forcent des changements d'état, et les modèles artificiels disposant de peu d'états sûrs, aucune stratégie sûre n'est permissive. Le temps d'exécution est reporté dans la troisième colonne. Sa croissance est raisonnable comparée à la différence d'ordre de grandeur de l'espace de recherche. La SCM n'a pas terminé pour le modèle `4var_4I`.

Ces résultats permettent d'apprécier les différences entre la SCM et la SEM. Comme prévu par la théorie, la SCM retourne plus de solutions et est plus lente que la SEM. La SEM est beaucoup plus rapide et assure la minimalité des résultats. Réduisant d'un facteur 10 le nombre de solutions de `3var_6`, elle apporte un gain non-négligeable lors de l'analyse des solutions et du choix d'une stratégie. Le seul inconvénient de la SEM est qu'elle n'est pas complète et ne permet donc pas de conclure à l'inexistence d'une solution. La synthèse des modèles de type `A` montre que la SCM est, dans ce cas, très rapide.

Il peut être surprenant qu'un modèle avec seulement quelques variables puisse requérir une synthèse de l'ordre de la demi-heure. Cependant, cela ne rend pas la synthèse inapplicable dans les cas réels. Premièrement, le nombre de variables qu'on utilise ici n'est pas irréaliste, un invariant de sécurité ne modélisant qu'un aspect dangereux du système. Le système réel étudié dans [Mekki-Mokhtar 2012a], donne lieu à des invariants de sécurité modélisés par une ou deux variables. Lors de notre étude de cas (c.f. chapitre 5), nous n'avons pas modélisé d'invariant avec plus de 4 variables. Deuxièmement, les modèles artificiels n'ont donc pas de dépendances entre variables. Les solutions à trouver sont donc plus nombreuses, quand il y en a, que dans des modèles réels. Ainsi, pour les modèle de l'étude de cas, les temps d'exécution de la SCM ont été inférieurs à la seconde.

Notre approche utilisant NuSMV pour la modélisation et la vérification a abouti au développement d'un outil de synthèse, performant pour les problèmes visés. S'appuyant sur un vérificateur existant, l'algorithme de coupe utilisé est spécifiquement conçu pour notre problème. Nous envisageons maintenant l'approche de la théorie des jeux, dotés d'outils génériques pouvant théoriquement traiter notre problème.

4.5 Ouverture sur la théorie des jeux

De manière très générale, nous avons jusqu'ici considéré un problème de vérification : étant donnés un comportement et des interventions, on cherche une

stratégie qui vérifie des propriétés (sécurité, permissivité). Or, en prenant du recul, notre problématique peut se formuler autrement : on cherche une stratégie telle que, *quoi que fasse* le système, il ne puisse pas atteindre les états catastrophiques (mais puisse atteindre tous les états non-catastrophiques). Cette formulation amène à considérer le problème sous la forme d'un jeu.

4.5.1 Approche par les jeux

Deux joueurs jouent donc l'un contre l'autre avec des moyens et des conditions de victoire différents, voire opposés. Chacun essaie d'atteindre sont but malgré le comportement du joueur adverse. Le moniteur, dont on adoptera le point de vue dans la suite, joue en exécutant sa stratégie. Sa condition de victoire est la sécurité et la permissivité. L'adversaire du moniteur est formé par la commande du système et l'environnement, qui peuvent mener le système dans des états catastrophiques. Le comportement de l'adversaire est non-déterministe. Dans le cas d'un jeu, on s'intéresse en particulier aux comportements correspondant à des fautes, par exemple tirer des transitions qui mènent dans un état catastrophique.

Au-delà de leurs conditions de victoire, les deux joueurs sont dissemblables par les moyens qu'ils ont d'influer sur l'état du système. En effet, l'adversaire peut tirer n'importe quelle transition alors que le moniteur ne commande que quelques transitions, dites *contrôlables*. On obtient donc un jeu sous la forme d'un automate, avec des formules de CTL comme conditions de victoire. Ce type de jeux est connu et courant, des outils permettent de les traiter.

La contrôlabilité des transitions rapproche notre jeu de la synthèse de superviseur de Wonham ([Wonham 2005], nous en donnons une présentation succincte dans [Machin 2013]). Cette méthode dispose d'un outil de synthèse, Supremica [Akesson 2006], mais ne prend en compte la permissivité telle que nous l'avons définie.

Des formalismes spécifiques aux jeux ont également été développés, comme par exemple la logique ATL (*Alternating Time Logic*, [Alur 2002]). Elle permet d'exprimer la condition de victoire d'un jeu indépendamment du point de vue d'un joueur, et donc permet de considérer la collaboration entre les joueurs pour réaliser une formule. Considérons un ensemble de joueurs Σ. Dans notre cas, on a $\Sigma = \{M, CE\}$, avec M, le moniteur, et CE, l'adversaire, c'est-à-dire la commande et l'environnement. Les opérateurs d'ATL sont obtenus en généralisant les quantificateurs de branche de la CTL par l'opérateur de «coopération» $\langle\langle\rangle\rangle$. Le quantificateur de CTL E (il existe une branche) est équivalent à $\langle\langle\Sigma\rangle\rangle$: si tous les joueurs coopèrent, il existe un chemin vérifiant la propriété. Au contraire, le quantificateur A (pour toutes les branches) est équivalent à $\langle\langle\emptyset\rangle\rangle$: si toutes les branches vérifient une propriété, il n'y a besoin de la coopération d'aucun joueur pour l'assurer. Notre problème pourrait donc être formulé en ATL. La sécurité étant la propriété $\langle\langle M\rangle\rangle G\neg cata$ et l'atteignabilité universelle $AG\Big(\langle\langle M, CE\rangle\rangle Fs_{nc}\Big)$. Cette logique permet donc d'exprimer la différence fondamentale entre la sécurité, qui doit être assurée par le moniteur *malgré* l'adversaire, et la permissivité pour laquelle les états ne

sont finalement atteints que par la coopération entre les deux joueurs. Dans un système multi-moniteurs, cette logique serait également utile pour déterminer les conditions de victoire et les coopérations de l'ensemble des joueurs. Cependant, ce formalisme n'est outillé à l'heure actuelle que pour la vérification (outil MOCHA [Alur 1998]) et non pour la synthèse.

4.5.2 Modélisation sous TIGA

A notre connaissance, le seul outil disponible publiquement permettant la synthèse à partir de CTL est TIGA [Behrmann 2007], l'extension pour la théorie des jeux d'UPPAAL, un outil de vérification de modèle présenté à la section 3.3.1. À part la contrôlabilité des transitions, les automates sont modélisés de la même façon que dans UPPAAL. La modélisation par variable n'est donc pas directement possible, chaque état doit être identifié explicitement. Par ailleurs, les propriétés de CTL acceptées sont réduites aux seules formes $AF\phi$ et $AG\phi$, avec ϕ une propriété de logique propositionnelle. Comme expliqué précédemment, ces propriétés ne sont pas suffisantes pour exprimer la permissivité. Cependant, nous avons réussi à encoder notre modèle, y compris la permissivité, au prix d'artifice de modélisation.

Nous nous proposons donc d'encoder dans TIGA le modèle défini à la section 3.2 avec trois restrictions :

— Les pré-conditions séquentielles ne sont pas modélisées.

— La permissivité considérée est une version bornée, donc plus forte, de l'atteignabilité universelle : tous les états non-catastrophiques doivent être atteignables en moins de k transitions depuis n'importe quel autre état non-catastrophique.

— Lorsqu'une action force le changement de valeur d'une variable, les transitions diagonales composées d'un forçage du moniteur et d'un changement de valeur voulu par l'adversaire ne sont pas possibles car les joueurs jouent à tour de rôle.

Par rapport à l'encodage en NuSMV, la modélisation pour TIGA doit être adaptée pour contourner deux obstacles : 1) faire une modélisation par variable alors que l'outil est pensé pour les modélisation par états, et 2) modéliser la permissivité dans l'automate puisque les conditions de victoire ne peuvent pas la prendre en compte.

Afin de modéliser par variable, les variables d'état de notre modèle sont encodées sans correspondances avec les états de TIGA : tous les états de notre automate son regroupés en un seul état TIGA, nommé `Situation` sur la figure 4.9. La transition nommée `Évolution` dans la figure 4.9 modélise les évolutions normales des variables d'état, dues à la commande principale et à l'environnement. Cette transition est donc incontrôlable. Lorsqu'`Évolution` est franchie, une nouvelle valeur est sélectionnée pour chaque variable (en respectant la continuité des variables). Chaque intervention du moniteur donne lieu à un franchissement de la transition nommée `Interv`. En cas d'action, la transition modifie les variables d'états, en cas

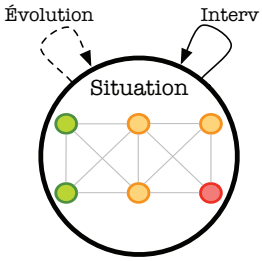


FIGURE 4.9 – Modélisation par variable

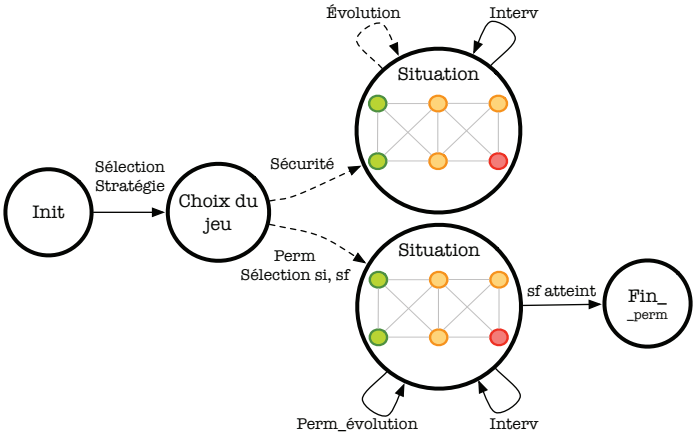


FIGURE 4.10 – Modélisation pour la permissivité

d'inhibition, la transition ajoute une garde temporaire sur la transition `Évolution`. Accompagné de la condition de victoire $AG\neg cata$, l'automate de la figure 4.9 permet de synthétiser une stratégie sûre, avec une modélisation par variables.

Cependant, les stratégies synthétisées n'offrent aucune garantie de permissivité. Comme les conditions de victoire ne peuvent pas exprimer la permissivité, on va la prendre en compte en modifiant l'automate valable pour la sécurité de la figure 4.9. L'automate comprendra donc deux jeux, un, tel quel, pour la sécurité, et un jeu pour la permissivité. Le choix du jeu se fait au départ. Dans le modèle TIGA de la figure 4.10, un état `Choix du jeu` est ajouté, d'où sortent deux transitions qui correspondent aux deux jeux. La stratégie attendue doit être sûre *et* permissive, elle ne doit pas dépendre du choix du jeu. Les transitions de choix sont donc incontrôlables et la sélection de la stratégie se fait avant le choix du jeu.

Modélisons maintenant la permissivité. Il faut vérifier l'existence d'un chemin entre chaque paire d'état `si`, `sf`. Cette paire est choisie par l'adversaire au départ du jeu pour que la stratégie soit valable pour n'importe quelle paire. Faire parcourir de tels chemins au système est du ressort de la commande. Or, ici, on requiert que ce soit fait par le joueur principal, qui représente le moniteur. Dans le cas d'un objectif de permissivité, il faut donc donner au joueur principal les moyens de déclencher des évolutions normales. La transition incontrôlable `Évolution` est donc remplacée, pour le jeu de permissivité, en une transition contrôlable `Perm_évolution`. Par ailleurs, la transition `Perm_évolution` n'inclut pas les transitions diagonales,

comme la vérification de la permissivité définie à la section 3.2.4 l'impose. L'objectif de permissivité est considéré comme rempli lorsque l'état final choisi `sf` est atteint. Le jeu est alors stoppé dans l'état `Fin_perm`.

La condition de victoire est $AG(\neg cata \wedge perm \rightarrow pas < k)$. Dans le cas où la sécurité a été choisie comme objectif, la condition de victoire reste inchangée par rapport au modèle pour la seule sécurité. En revanche, pour la permissivité, le nombre de transitions franchies *pas* ne doit pas dépasser k, c'est-à-dire que l'état `Fin_perm`, donc `sf`, doit être atteint avant.

4.5.3 Résultats et comparaison

Nous avons donc réussi à encoder notre modèle, à quelques restrictions près, dans TIGA. L'outil synthétise une stratégie qui est effectivement correcte. Cependant, les performances de l'outil sont très mauvaises pour nos modèles, comme détaillé dans [Machin 2015a]. La synthèse échoue faute de mémoire pour un modèle avec 2 variables à 3 valeurs chacune, et 4 interventions alors que pour le même modèle et dans les mêmes conditions d'exécution, la version non-parallélisée de notre outil de synthèse met 1,3s à trouver toutes les stratégies minimales (la variante SEM est utilisée car il n'y a pas de pré-condition séquentielle dans ce modèle). Les performances décevantes de TIGA s'expliquent par le fait que sa modélisation n'est pas adaptée à notre problème. En particulier, exprimer la permissivité nécessite d'ajouter de nombreux artefacts de modélisation et de considérer la stratégie comme une variable d'état.

Notre algorithme de synthèse, spécifiquement conçu pour nos besoins, est donc plus performant que l'outil générique TIGA, et la modélisation associée est plus directe et lisible. Nous avons démontré que notre modélisation est encodable dans d'autres outils que NuSMV, et que la synthèse peut se faire avec une approche différente de la vérification de modèle.

4.6 Conclusion

Nous avons montré que la modélisation choisie au chapitre 3 permet la définition d'un algorithme de synthèse. Définies à partir des états d'alerte et des interventions, les stratégies sont structurées dans un arbre, ce qui permet de les construire graduellement. L'exploration de l'arbre à la recherche des stratégies correctes et minimales est réduite par des critères de coupe, dont nous avons étudié les propriétés de complétude et d'exclusivité. Les tests ont montré que les critères de coupe sont efficaces en pourcentage de nœuds explorés. Nous avons mis en œuvre l'algorithme de parcours d'arbre dans un outil, qui fait appel à NuSMV pour la vérification des critères de coupe. Ses performances sont satisfaisantes pour le type de problèmes traités.

Notre problématique peut être abordée en utilisant la théorie des jeux. Si l'encodage ne permet pas actuellement d'obtenir des performances satisfaisantes, les

bases formelles de modélisation sont les mêmes. Ainsi, notre problème de synthèse pourra trouver des solutions plus performantes, à mesure que les outils des méthodes formelles s'amélioreront. Pour l'heure, les performances de notre outil sont suffisantes pour nous permettre de traiter une étude de cas.

Ce qu'il faut retenir :

- Les propriétés d'exclusivité et de complétude des critères de coupe définis sont connues.
- Les critères de coupe sont efficaces.
- L'outil développé permet de conclure rapidement à l'inexistence de solutions.
- Les bases formelles de la modélisation sont indépendantes de l'approche de synthèse. Deux approches de synthèse sont proposées : la vérification et la théorie des jeux.

Chapitre 5

Étude d'un cas industriel

Sommaire

Pour illustrer notre méthode, nous l'appliquons sur un cas d'étude fourni par Kuka [1], un important fabricant de robots manufacturiers, dans le cadre du projet européen Saphari [2] (*Safe and Autonomous Physical Human-Aware Robot Interactions*, FP7). Le système est présenté par la section 5.1. Cette étude de cas a permis d'appliquer notre méthode et de tester le moniteur ainsi spécifié. Les invariants ont été obtenus par une analyse de risque Hazop-UML, succinctement présentée par la section 5.2. Puis, la section 5.3 présente la construction du modèle, la synthèse et l'analyse de conflit qui ont permis de déterminer un ensemble non-conflictuel de stratégies sûres et permissives. La section 5.4 montre comment cet ensemble a été implémenté et testé dans le système réel.

5.1 Présentation du robot

Le système sur lequel nous appliquons notre méthode est le robot manufacturier OmniRob, conçu pour évoluer dans l'espace intérieur d'une usine et côtoyer des

1. `www.kuka.com`, consulté le 7 juillet 2015
2. `www.saphari.eu`, consulté le 7 juillet 2015

FIGURE 5.1 – Le robot mobile manipulateur Omnirob de KUKA

opérateurs qualifiés sans barrières de protection. Le système, représenté en figure 5.1 est composé d'une plate-forme mobile avec quatre roues omni-directionnelles et d'un bras articulé à sept degrés de liberté de type LWR (*Lightweight Robot*). Le dimensionnement et la commande du robot OmniRob ont été réalisés tels qu'il puisse partager son espace, voire collaborer avec des opérateurs humains. Ainsi, il pourrait participer rapidement, et idéalement, sans programmation préalable, à la production de petites séries, de prototypes.

Dans le cas d'étude présent, la tâche choisie est la suivante : le robot doit approvisionner des postes de travail en pièces détachées, contenues dans des boîtes. Elles sont saisies, une par une, par la pince dont est équipé le bras du robot, et posées sur le haut de la plate-forme pour être transportées. Elles sont ensuite posées sur les tables qui constituent les postes de travail. Le robot peut également être utilisé pour faciliter l'inspection d'une pièce lourde par l'opérateur. La pièce est saisie par la pince, et le bras est guidé physiquement par l'opérateur. Le bras tient la position, ce qui permet à l'opérateur de réaliser l'inspection sans avoir à fournir d'effort physique. Le début et la fin de l'interaction sont marqués par des commandes dites haptiques : l'opérateur imprime au bras un mouvement prédéfini. Ce mouvement doit être assez inhabituel pour pouvoir être discriminé des autres interactions physiques. Dans notre cas, le mouvement envisagé est de pousser le «coude» du robot. Par ailleurs, le cas d'étude spécifie une zone de l'usine comme interdite au robot.

La commande du bras, telle que développée par KUKA, inclut une «couche de sécurité», c'est-à-dire un contrôle séparé et redondé, satisfaisant la plupart de nos hypothèses sur le moniteur de sécurité. L'interface du moniteur est définie en terme de variables observables et d'interventions. Le comportement entrée/sortie doit être spécifié sous la forme d'associations d'interventions et de formules de logique propositionnelle sur des prédicats de variables observables. Les formules doivent être exprimées dans la forme normale disjonctive ($\bigvee \bigwedge$). Ce cadre convient à nos stratégies car les interventions sont associées à une union d'états, chacune étant représentée par une conjonction de valeur des variables. En outre, la forme des

prédicats utilisables dans le moniteur implémenté est réduite à une comparaison à une constante.

Dans le système, seules deux interventions sont possibles : le freinage du bras et le freinage de la plate-forme. Les observations utilisées seront détaillées au fur et à mesure de la modélisation des invariants dans la section 5.3.

5.2 Analyse de risque et formulation des invariants

Notre étude de cas débute par l'analyse de risque Hazop-UML des cas d'utilisations du système. Au regard de la polyvalence et de la diversité des environnements, mener une étude sécurité à partir des cas prévus d'utilisations peut paraître trop restreint. Cependant, ces cas sont un bon support pour définir des scénarios alternatifs et imaginer les détournements et les mésusages du système.

L'analyse de risque Hazop-UML a été présentée à la section 1.2. Le système est d'abord modélisé en UML par 11 cas d'utilisation, représentés par 50 attributs et 15 diagrammes de séquence. Puis, l'analyse Hazop de ces éléments est conduite, et reportée dans les tables Hazop. Une centaine de lignes ont une gravité non-nulle. Chaque ligne est censée être modélisée par un invariant de sécurité. En pratique, après avoir fait l'analyse préliminaire de sécurité et l'Hazop, la connaissance du système est telle que les lignes d'Hazop similaires sont regroupées rapidement. Treize invariants de sécurité ont été formulés et sont détaillés dans le rapport [Machin 2015b]. La liste de ces invariants est donnée ci-dessous :

SI1	La vitesse du bras doit être inférieure à V_1.
SI2	La vitesse de la plate-forme doit être inférieure à V_2.
SI3	Le robot ne doit pas entrer dans la zone interdite.
SI4	La plate-forme ne doit pas entrer en collision avec un être humain.
SI5	Le bras ne doit pas être étendu au-delà de la plate-forme lorsque la plate-forme bouge.
SI6	Une boîte saisie par la pince ne doit pas être penchée de plus de α_0.
SI7	Une collision entre le bras du robot et un être humain ne doit pas blesser l'être humain.
SI8	La vitesse de tout point du robot ne doit pas dépasser V_1.
SI9	Les boîtes ne doivent pas être lâchées.
SI10	Le bras ne doit pas écraser un être humain.
SI11	La pince ne doit pas écraser un être humain (en particulier ses doigts).
SI12	Le robot ne doit pas faire basculer, en les heurtant, les boîtes disposées sur le haut de la plate-forme, les tables.
SI13	Le robot doit suivre le guidage physique.

5.3 Modélisation et synthèse

À partir de ces invariants, des observations et interventions du moniteur, la méthode est appliquée. Pour chaque invariant, un modèle est construit et encodé

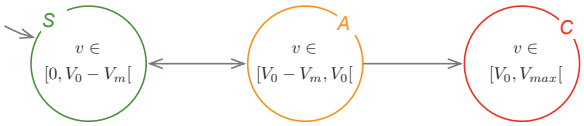


FIGURE 5.2 – Comportement du modèle pour l'invariant SI1

comme présenté aux sections 3.2 et 3.3. La synthèse est effectuée et les résultats sont discutés (section 3.4). Après avoir choisi une stratégie pour chaque invariant, l'analyse des conflits (section 3.5) entre stratégies est conduite.

5.3.1 Les invariants simples (SI1 à SI4)

Le premier invariant est la limite de vitesse du bras, formalisé par $v_b < V_0$. La seule variable observable est la vitesse du bras, elle admet une marge. Ainsi, le comportement n'a que 3 états, comme représenté sur la figure 5.2. Le freinage du bras est la seule intervention modélisable (son encodage correspond à la modélisation donnée par le tableau 3.5). L'encodage du modèle se réduit à :

```
Continuity(max_value, init_value)
VAR v : Continuity(2,0);
DEFINE cata := v=2;
VAR brake : Intervention(TRUE, v=0,flag_brake, next(v)!=2);
```

Une seule stratégie est synthétisée, et, comme attendu, le freinage est associé à l'état d'alerte A, dans lequel `v=1`.

Le modèle et la stratégie sont simples. Cependant, l'implémentation soulève des questions. En effet, définir la vitesse d'un bras à 7 axes peut se révéler complexe. La variable observable par le moniteur implémenté est la vitesse maximale parmi les vitesses (translationnelles) des points centraux des articulations du bras exprimées dans le repère cartésien. (Le seuil de sécurité V_0 et la marge ont été déterminés par les ingénieurs de KUKA). La vitesse du bras est exprimée dans le référentiel de la base du bras, c'est-à-dire le référentiel de la plate-forme, qui est bien sûr différent du référentiel terrestre/de l'environnement. En conséquence, un autre invariant (SI8) est défini pour limiter la vitesse du robot, ou plus exactement de l'ensemble de ses points, dans le référentiel terrestre. Intuitivement, la somme de la vitesse du bras et de la plate-forme doit être limitée. Toutefois, la somme vectorielle ne fait pas partie des observations disponibles.

Le modèle pour la limitation de la vitesse de la plate-forme (SI2) est identique à celui du SI1.

Le troisième invariant, qui stipule que le système ne doit pas entrer dans la zone interdite, a un modèle similaire. Le modèle ne comprend qu'une intervention, le freinage de la plate-forme, et une observation, la différence entre la distance à la

zone interdite (calculée par localisation dans la carte) et la distance de freinage (calculée à partir de la vitesse). Le modèle utilise donc une variable qui n'est pas directement observable mais calculée à partir de variables observables. Le calcul est suffisamment simple pour être effectué par le moniteur de sécurité.

De même, la collision de la plate-forme (SI4) est évitée en observant la différence entre la distance de freinage et la distance à l'obstacle.

5.3.2 SI5 : *Le bras ne doit pas être étendu au-delà de la plateforme lorsque la plate-forme bouge.*

L'exemple introduit à la section 3.1 est inspiré de cet invariant. Toutefois pour des raisons de représentatibilité et de simplicité, la formulation de l'invariant de l'exemple a été légèrement modifiée par rapport à l'invariant du cas d'étude. On verra que cette modification de formulation induit une grande différence dans le modèle et la stratégie choisie.

Dans la présente formulation «*Le bras ne doit pas être étendu au-delà de la plate-forme lorsque la plate-forme bouge*», l'observation de la plate-forme `pf_vel` est un booléen qui correspond à un mouvement (`1`) ou une absence de mouvement (`0`). La vitesse, variable observable, est utilisée avec un seuil très bas pour observer le mouvement.

Le moniteur implémenté permet de définir comme un observable le fait que le bras soit dans un certain espace défini par l'utilisateur. Si tous les points du bras sont dans l'espace, celui-ci est respecté ; sinon il est enfreint. Cette observation sous forme d'espace peut donc être vue comme un prédicat en 3 dimensions sur la position du bras. De la même manière que les autres prédicats sont limités à la comparaison à une constante, l'espace est défini statiquement par un parallélépipède rectangle attaché au référentiel de la base du bras (équivalent à celui de la plate-forme). Pour exprimer notre invariant, on utilise cette observation propre au moniteur implémenté.

La limite exprimée par l'invariant correspond à l'empreinte de la plate-forme. Pour l'observer, nous définissons un espace dit *espace extérieur*, dessiné sur la figure 5.3. La marge consiste à prendre un espace plus petit, inclus dans l'empreinte de la plate-forme. Sur la figure 5.3, un deuxième espace est donc défini, qui est appelé *espace intérieur*. La variable `arm_pos` modélisant la position du bras est définie à partir de ces deux espaces par le tableau 5.1. Par exemple, la position du bras représentée dans la figure 5.3 est encodée par `arm_pos=1`.

Pour cet invariant, les freinages du bras et de la plate-forme sont tous les deux pertinents et modélisables. La marge de l'extension du bras, c'est-à-dire l'écart entre les deux espaces de travail, est calculée en fonction de la distance de freinage du bras. Le freinage du bras `brake_arm` est toujours valide et, s'il est déclenché depuis l'espace intérieur (`arm_pos=0`), il empêche le bras de sortir de l'espace extérieur, c'est-à-dire qu'il rend impossible la valeur `2`. Si la marge (`arm_pos=1`) est atteinte après l'application du freinage, le bras sera stoppé dans la marge, et ne pourra donc plus atteindre ni l'espace intérieur (`arm_pos=0`), ni enfreindre l'espace

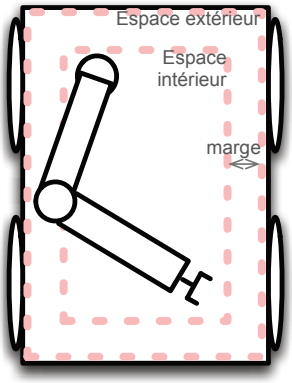


FIGURE 5.3 – Définition des espaces pour l'observation de la position du bras sur une vue de haut du robot. Dans la position représentée, l'espace extérieur est respecté et l'espace intérieur est enfreint.

Extérieur \ Intérieur	respecté	enfreint
respecté	0	1
enfreint	-	2

TABLEAU 5.1 – Valeurs de la position du bras (variable `arm_pos`)

extérieur (`arm_pos=2`). Le freinage du bras ainsi que celui de la plate-forme sont modélisés par :

```
Intervention(stat_precond, seq_precond, condition, effect)
        next(arm_pos)=1 | next(arm_pos)=arm_pos);
brake_pf : Interv( pf_vel=0, TRUE , flag_brake_pf,
        next(pf_vel)=pf_vel);
```

Le freinage de la plate-forme `brake_pf` empêche son démarrage lorsqu'elle est à l'arrêt. Étant donnée l'inertie de la plate-forme et la puissance de freinage, nous faisons l'hypothèse que le freinage est instantanément efficace, c'est-à-dire que la vitesse de la plate-forme ne peut pas atteindre une vitesse significative entre l'instant où le freinage est appliqué par la commande et l'instant où les freins sont physiquement enclenchés. Cette hypothèse est validée par les ingénieurs de KUKA. En conséquence, le freinage modélisé n'a pas de pré-condition séquentielle, ou en d'autres termes, sa pré-condition séquentielle est toujours vraie. L'effet est modélisé en s'appuyant sur le fait que la plate-forme est à l'arrêt, ce qui donne lieu à la pré-condition statique `pf_vel=0`.

La stratégie synthétisée est la suivante :

```
DEFINE flag_brake_arm := pf_vel=1 & arm_pos=1 ;
DEFINE flag_brake_pf := (pf_vel=0 & arm_pos=1) | (pf_vel=0 & arm_pos
    =2) ;
```

Le freinage de la plate-forme `brake_pf` est associé aux états tels que la plate-forme est à l'arrêt et le bras est près ou au-delà de la limite de l'empreinte de la plate-forme. Le freinage du bras `brake_arm` est associé aux états tels que la plate-forme

est en mouvement et le bras est dans la marge.

5.3.3 SI6 : *Une boîte saisie par la pince ne doit pas être penchée avec un angle supérieur à* α_0

Si le bras est équipé d'une pince, il est capable de saisir des boîtes contenant des pièces. Comme les boîtes ne sont pas fermées, les pièces en tombent lorsqu'elles sont trop penchées.

Pour formaliser cet invariant, les observations nécessaires sont 1) la présence d'une boîte dans la pince, notée *box*, et 2) l'angle maximum de la boîte par rapport aux axes x et y [3] du repère cartésien en valeur absolue, noté α. Étant donné un seuil d'angle α_0, l'invariant est formalisé par $\alpha < \alpha_0 \vee box = faux$.

L'invariant est formulé avec l'angle de la boîte dans le repère cartésien. Or, cet angle n'est observable que par l'intermédiaire de l'angle de la pince. Ils sont identiques sous l'hypothèse que les boîtes sont toujours saisies par le haut. Étant données la forme et l'emplacement des encoches de la boîte permettant de la saisir, cette hypothèse est réalisée. L'angle défini est observable et admet une marge. On le note `alpha`, et on définit la partition `alpha=0` si $\alpha < \alpha_0 - \alpha_m$, `alpha=1` si $\alpha_0 - \alpha_m \leq \alpha < \alpha_0$ et `alpha=2` si $\alpha \geq \alpha_0$. La marge α_m est dimensionnée par la distance de freinage (en angle) du bras. L'observation de la présence d'une boîte n'est pas observable par le moniteur.

Maintenant que toutes les observations nécessaires sont disponibles, on peut définir la sécurité dans le modèle en posant `cata := alpha=2 & box=1`. La synthèse ne retourne aucune solution. Le moniteur est en effet incapable d'empêcher la transition de l'état d'alerte `alpha=2 & box=0` à `cata`. Cependant, étant donnée la forme de la pince, la saisie se fait toujours avec la boîte à l'horizontale. Ainsi, lorsque la variable `box` passe à `1`, l'angle de la pince est toujours inférieur à α_0, c'est-à-dire `alpha=0`. On peut donc ajouter au modèle la dépendance :

```
|| TRANS box=0 & next(box)=1 -> alpha=0
```

La synthèse retourne alors une solution qui associe le freinage du bras à l'état `box=1 & alpha=1`, c'est-à-dire quand le bras a saisi une boîte et la penche.

5.3.4 SI7 : *Une collision entre le bras du robot et un être humain ne doit pas blesser l'être humain*

La collision est un danger typique en robotique. Une première approche est l'évitement de collision : par exemple le SI4 interdit la collision entre un être humain et la plate-forme. Cependant, la collision peut être une interaction physique tolérée si le système a une inertie faible et est équipé pour limiter la gravité de la collision.

Tolérer les collisions suppose de pouvoir discriminer les collisions inoffensives des collisions dangereuses. Un premier paramètre est la vitesse à laquelle la collision a lieu. Un second est la dynamique de la collision, c'est-à-dire le couple

3. Une rotation autour de l'axe z ne penche pas la boîte, et ne risque donc pas de faire tomber les pièces.

Observation	Notation	Parties	Seuils définissant les parties
Couple externe	`ext_torq`	`0-1`	Seuil de collision (norme)
Vitesse du bras	`arm_vel`	`0-1`	Seuil de vitesse (norme)
Position de la pince	`grip_pos`	`0-1`	Une boîte peut être saisie (`0`) ou pas (`1`)
Position de la plate-forme	`pf_pos`	`0-1`	La plate-forme est proche des stockages (`0`) ou pas (`1`)

Tableau 5.2 – Observations directes utilisées pour couvrir la collision (SI7)

de collision, difficile à observer en pratique. La couche de sécurité Kuka dispose du *couple externe* comme observation. Le couple externe est calculé à partir des couples mesurés au niveau des articulations et du modèle du bras (géométrie et masse, couples internes produits par le poids et l'accélération). Quand une boîte est saisie par la pince, la boîte est considérée comme solidaire du bras. Les couples que son poids et son inertie génèrent ne doivent pas être considérés comme faisant partie du couple externe. Il faut donc modifier le modèle de calcul pour obtenir un couple externe correct. La modification du modèle se fait à l'initiative de la commande, un mécanisme du moniteur vérifiant que l'objet ajouté au modèle et correspond à l'objet réellement saisi. Sur le principe, ce mécanisme passe par la mesure du couple externe dans des conditions telles qu'il est seulement constitué du couple dû à la boîte. Durant un certain laps de temps, le modèle, et donc l'observation du couple externe, ne sont donc pas valides.

Pour s'affranchir de ce problème, nous avons choisi d'exclure de l'invariant les cas dans lesquels le modèle est appelé à changer. L'observation du couple externe est donc accompagnée de conditions excluant les situations de changement, qui sont approximées grossièrement par la position de la plate-forme dans l'atelier et la position de la pince par rapport à la hauteur des stockages (étagères, tables, haut de la plate-forme). Ces situations, exclues de cet invariant, devront être couvertes par un autre invariant. Du point de vue de la permissivité, la séparation se justifie, car dans ces cas, un couple externe est attendu, et même désiré, pour placer les boîtes sur les stockages. Du point de vue de la sécurité, il est courant en robotique, y compris dans les normes, de séparer le traitement de la collision dans un espace libre et de celui de l'écrasement contre un obstacle.

Les observations directes utilisées pour détecter une collision dangereuse sont récapitulées dans le tableau 5.2. La collision est donc une observation indirecte définie comme suit :

```
INVAR collision=1 <->
        (ext_torq=1 & grip_pos=1 & pf_pos=1 & arm_vel=1)
```

La collision ne causant de dommages que lorsqu'elle dure un certain temps, on introduit un timer sur la collision, une observation indirecte calculée par le moniteur lui-même. On définit donc la variable `col_duration`, qui permet de définir les états catastrophiques et admet une marge.

```
ASSIGN init(col_duration):=0;
next(col_duration):=case
```

```
  next(collision)=1 & col_duration=0 :1; --timer launches
  next(collision)=1 & col_duration=1 :2; --timer expires
  col_duration=2  :2;
  TRUE  :0;
DEFINE cata:= col_duration=2;
```

La synthèse retourne une stratégie qui freine le bras dès que le timer est lancé.

```
flag_brake_arm := ext_torq=1 & arm_vel=1 & grip_pos=1 & pf_obs=1 &
    col_duration=1 ;
```

5.3.5 Les autres invariants (SI8 à SI13)

Ces invariants ne peuvent pas être couverts par le moniteur du robot. La somme vectorielle des vitesses du bras et de la plate-forme requise pour le SI8 n'est ni modélisable, ni observable, par la couche de sécurité. Pour empêcher les boîtes d'être lâchées (SI9), l'intervention de blocage de la pince manque.

L'écrasement provoqué par le bras (SI10) est observé par les forces externes au niveau de la pince. Cependant, après modélisation, synthèse et expérimentation, les valeurs des seuils et des marges ne permettent d'assurer la sécurité qu'à un prix exorbitant en permissivité : réduction drastique de la vitesse, environnement mou pour obtenir une croissance observable des forces. En conséquence, le modèle et la stratégie ne sont pas pris en compte ici. L'écrasement par la pince (SI11) est couvert par le choix de la pince, intervenu une fois l'analyse de risque faite. Pour saisir les boîtes, l'outil choisi n'est pas une pince dont les doigts sont actionnés mais un effecteur non-actionné dont la forme correspond à une encoche sur les boîtes qui permet de les soulever.

Éviter que la pince ne heurte et fasse basculer les boîtes (SI12) nécessite un modèle complexe incluant le mouvement de la plate-forme, le mouvement du bras par rapport à l'environnement et les positions des boîtes. De telles observations ne sont pas disponibles et on peut douter de la confiance à placer dans un modèle aussi complexe, ainsi que dans le moniteur qui en résulterait. Pour vérifier que le guidage physique du bras (SI13) est bien suivi, il faudrait comparer les mouvements et les couples externes sur les 7 axes du bras, ce qui amènerait beaucoup de calculs. Une approche moins permissive serait d'établir des restrictions, par exemple de vitesse, d'accélération, lorsque le robot est guidé manuellement. Le moniteur doit pour cela être capable d'observer le début et la fin de cette phase. Or, le protocole d'interaction pour commencer et terminer cette phase est constitué de commandes haptiques. Elles peuvent être difficilement observables par le moniteur, et surtout, on ne peut pas les différencier avec certitude d'interactions physiques intentionnelles.

5.3.6 Analyse de conflit

Après avoir réuni les instances du patron définies pour les invariants 1 à 7, avec leurs stratégies, les observations et les interventions similaires sont mises en cohérence. La vitesse du bras est utilisée dans SI1 et SI7 avec des partitions

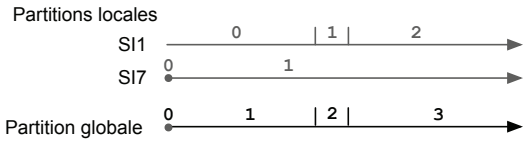


FIGURE 5.4 – Mise en cohérence des partitions utilisées pour la vitesse du bras

différentes, la cohérence est assurée par la partition globale définie en figure 5.4. De même pour la vitesse de la plate-forme utilisée par SI2 et SI5. Les déclenchements du freinage du bras d'une part, et les déclenchements du freinage de la plate-forme d'autre part sont synchronisés.

L'analyse de conflit a d'abord pour but de vérifier la non-concomitance d'interventions incompatibles. Ici, les deux interventions utilisées ne sont pas incompatibles. La propriété de non-concomitance est fausse sans que cela soulève de problème. Dans un second temps, le modèle global peut permettre de révéler des pertes de permissivité, la combinaison des stratégies pouvant rendre fausses des propriétés de permissivité vérifiées sur les instances de patron. Après vérification, les propriétés locales de permissivité sont toutes vraies.

5.4 Implémentation et tests des stratégies

Afin de tester notre approche de modélisation et de synthèse, nous avons implémenté les stratégies obtenues pour SI1, SI2, SI3, SI5 et SI7.

5.4.1 Conditions d'expérimentation

Les tests que nous conduisons sont faits sur le système réel dans l'environnement du laboratoire de KUKA. Le robot mis à notre disposition n'était pas équipé de pince, différant en cela du robot du cas d'étude théorique. C'est pourquoi la stratégie de SI6 n'a pas pu être implémentée. La stratégie de SI4 n'a pas été implémentée au vu de sa similarité avec SI3 et de la dangerosité des tests, impliquant des collisions entre la plate-forme et des obstacles. Lors des tests, nous n'avons pas pu avoir accès à la couche de sécurité présente sur le système. Les stratégies sont donc implémentées par l'interface Java haut-niveau du système. Les hypothèses de correction du moniteur de sécurité ne sont clairement pas satisfaites, notamment, les temps d'échantillonnage des observations et les temps de latence d'envoi de requêtes d'interventions sont inconnus. En conséquence, nous ne pouvons pas valider ce moniteur ou ces stratégies. Toutefois, nos expériences démontrent l'applicabilité de la méthode et la faisabilité de l'implémentation des stratégies.

5.4.2 Cas de test

Afin d'évaluer les stratégies implémentées, nous devons tester le moniteur, donc en particulier déterminer des cas de test. Nous définissons en premier lieu

une mission : à partir d'une position initiale déterminée, une position finale doit être atteinte en un certain temps. Le cas de référence, qui correspond à un mouvement générique du bras puis de la plate-forme, est déterminé pour réussir cette mission. Nos objectifs de tests haut-niveau sont la sécurité et la permissivité du moniteur. Les invariants étant le point de départ de notre méthode, ils servent de base pour définir des déviations du cas de référence qui forment les cas de tests. La première colonne du tableau 5.3 présente les cas de test et les objectifs associés à chacun. Le nombre peu important de cas est à relier aux conditions de test : les tests ont lieu sur le système réel et non en simulation, ils peuvent être dangereux. En outre, l'environnement de test est situé chez KUKA, en Allemagne.

Pour décorréler l'évaluation de la sécurité et la permissivité, la réussite de la mission est définie indépendamment de toute considération de sécurité. Les cas de test changent les conditions dans lesquelles la mission doit être accomplie, par exemple en plaçant un être humain à la position finale. Mais ils ne changent pas les critères de réussite de la mission : atteindre la position finale est une réussite de la mission, quand bien même cela implique que le robot entre en collision avec l'être humain.

La deuxième colonne du tableau 5.3 présente les résultats des exécutions des cas de test. Chaque cas de test a été exécuté 2 fois avec le moniteur et 2 fois sans le moniteur. Les exécutions jumelles ont eu des résultats similaires vis-à-vis des données mentionnées dans le tableau 5.3. Les interventions appliquées lors des exécutions sont mentionnées pour chaque exécution (colonnes Freinage bras et Freinage plate-forme). Si aucun état catastrophique du modèle n'est atteint, c'est-à-dire si aucun invariant n'a été violé (colonne Invariant violé), l'exécution est dite sûre. Une évaluation visuelle de la sécurité est également faite pour couvrir les dangers non-modélisés. Lors de nos expériences, cette évaluation visuelle a toujours concordé avec la sécurité modélisée par les invariants. Une exécution réussit la mission (colonne Mission réussie) si les positions finales prévues du bras et de la plate-forme sont atteintes dans la durée spécifiée.

À partir des résultats des exécutions, le comportement du moniteur est évalué dans chaque cas de test (troisième colonne du tableau 5.3). Un cas est pertinent pour évaluer la permissivité du moniteur si toutes ses exécutions sont sûres et qu'en l'absence de moniteur la mission est réussie. Le moniteur est permissif (noté $\surd$ dans la colonne Permissivité) si la mission est réussie en sa présence. Un cas est pertinent pour la sécurité si au moins une de ses exécutions sans moniteur n'est pas sûre. Pour un cas pertinent, le moniteur est sûr (noté $\surd$) si toutes les exécutions avec moniteur sont sûres. Sinon, le comportement du moniteur n'est pas sûr (noté X). L'évaluation de la permissivité (respectivement de la sécurité) sur un cas non pertinent pour la permissivité (respectivement sécurité) est indiquée par la notation "NP".

5.4.3 Résultats

Nous discutons ici des résultats des expériences présentés dans le tableau 5.3. Dans le cas de référence (cas 0), les exécutions se déroulent sans porter atteinte à la sécurité et la mission est réussie. Ce cas ne permet pas de conclure sur la sécurité car aucun invariant n'est violé par les exécutions sans moniteur. En revanche, le moniteur est permissif vis-à-vis de ce cas.

Une requête de la commande comportant une accélération normale et une vitesse du bras supérieure au seuil de sécurité (cas 1) fait freiner le moniteur, ce qui empêche la violation du SI1 et montre la sécurité vis-à-vis du cas 1. En revanche, ce cas n'est pas pertinent pour l'évaluation de la permissivité puisque certaines exécutions ne sont pas sûres. Lorsqu'une accélération élevée et une vitesse dépassant le seuil sont commandées (cas 2), l'état catastrophique est atteint, même en présence du moniteur. Le freinage est appliqué comme prévu mais n'empêche pas l'état catastrophique d'être atteint (transitoirement). Le moniteur est donc non-sûr pour le cas 2. La valeur de marge est trop faible vis-à-vis du temps de latence entre la commande et l'effet physique des freins, ou d'un autre point de vue, elle nécessite une hypothèse d'accélération maximale qui n'est pas réalisée.

Les cas de test 1 et 2 portent sur la vitesse du bras, les cas 3 et 4 sur la vitesse de la plate-forme. Dans le cas 3 (respectivement le cas 4), les résultats mentionnés dans le tableau 5.3 sont les mêmes que le cas 1 (respectivement le cas 2), et amènent donc aux mêmes conclusions. Lors d'une exécution du cas 4 en l'absence de moniteur, on a par ailleurs observé un saut de la valeur de la vitesse de la plate-forme : $v_{pf} < V_1 - V_m$ puis au pas suivant $v_{pf} > V_1$. Cette évolution viole la continuité de la variable abstraite et n'existe donc pas dans le modèle. Au niveau des états du modèle, on passe d'un état sûr à un état catastrophique, sans que soit observé un état d'alerte. La valeur de la marge est donc trop faible par rapport à la dynamique du système et au temps de cycle. Ceci est principalement dû aux conditions d'expérimentation, en particulier à l'implémentation du moniteur dans une couche applicative et pas dans la couche de sécurité temps réel.

Pour le cas 5, la commande principale envoie une requête pour fixer la position finale de la plate-forme dans la zone interdite. Les exécutions sans moniteur violent un invariant mais, en présence du moniteur, le freinage de la plate-forme empêche d'atteindre la zone interdite et assure donc la sécurité.

La commande principale du cas 6 requiert du bras une position en-dehors de l'empreinte de la plate-forme, faisant donc systématiquement échouer la mission. Pendant le mouvement de la plate-forme, le bras est freiné avant d'être étendu au-delà de l'empreinte de la plate-forme, empêchant l'état catastrophique. Dans le cas 7, les freins de la plate-forme sont appliqués à l'arrêt et empêchent le démarrage du bras, assurant ainsi la sécurité. Les cas 8 et 9 n'amènent à aucune violation d'invariant, ils ne sont donc pas pertinents pour évaluer la sécurité. Dans le cas 9, les freins de la plate-forme sont enclenchés lorsque le bras sort de l'empreinte de la plate-forme. En effet, le moniteur garantit ainsi que, même si la commande demandait un démarrage de la plate-forme, il ne serait pas possible. La mission est

Définition des cas de test			Exécutions et résultats					Évaluation du moniteur	
Numéro	Description	Objectif de test	Moniteur	Freinage bras	Freinage plate-forme	Invariant violé	Mission réussie	Permissivité	Sécurité
0	Cas de référence : Mouvement du bras (au-dessus de la plate-forme) puis mouvement de la plate-forme.	Permissivité	Sans				√	√	NP
			Avec				√		
1	La commande requiert du bras une vitesse supérieure au seuil de sécurité, avec une accélération normale.	SI1	Sans			SI1	√	NP	√
			Avec	√					
2	La commande requiert du bras une vitesse supérieure au seuil de sécurité, avec une accélération élevée.	SI1	Sans			SI1	√	NP	X
			Avec	√		SI1			
3	La commande requiert de la plate-forme une vitesse supérieure au seuil de sécurité, avec une accélération normale.	SI2	Sans			SI2	√	NP	√
			Avec		√				
4	La commande requiert de la plate-forme une vitesse supérieure au seuil de sécurité, avec une accélération élevée.	SI2	Sans			SI2	√	NP	X
			Avec		√	SI2			
5	La commande requiert de la plate-forme d'aller dans la zone interdite.	SI3	Sans			SI3		NP	√
			Avec		√				
6	Pendant le mouvement de la plate-forme, le bras bouge au-delà de l'empreinte de la plate-forme.	SI4	Sans			SI5		NP	√
			Avec	√	√				
7	Pendant le mouvement du bras, le bras bouge au-delà de l'empreinte de la plate-forme et s'y arrête. La plate-forme bouge normalement.	SI5	Sans			SI5		NP	√
			Avec		√				
8	Pendant le mouvement de la plate-forme, le bras bouge au-dessus de la plate-forme.	Permissivité (SI5)	Sans				√	√	NP
			Avec				√		
9	Le bras bouge au-delà de l'empreinte de la plate-forme et s'arrête au-dessus de la plate-forme. Puis, la plate-forme bouge normalement.	Permissivité (SI5)	Sans				√	√	NP
			Avec		√		√		
10	Collision entre le bras et un humain, l'extrémité du bras étant dans l'espace libre.	SI7	Sans			SI7	√	NP	√
			Avec		√				

Tableau 5.3 – Définition des cas de test et résultats de leurs exécutions

réussie dans les exécutions des cas 8 et 9, y compris en présence du moniteur, ce qui permet de conclure à la permissivité du moniteur vis-à-vis de ces cas.

Le cas 10 est obtenu par une collision entre un être humain et le bras de robot. Comme la collision a lieu dans un espace libre, le risque d'écrasement par le bras est nul. Afin de rendre acceptable le niveau de risque de l'expérience, pour un opérateur expérimenté et attentif et dans des conditions de laboratoire, le mouvement de la plate-forme est désactivé. La position de la plate-forme n'est donc pas prise en compte dans la réussite de la mission. Les exécutions du cas 10 montrent que l'état catastrophique est évité par le freinage du bras. Le moniteur est donc sûr vis-à-vis du cas 10.

Les stratégies implémentées assurent partiellement la sécurité. Les expériences confirment une dépendance forte de la méthode au calcul des marges. Cela requiert une expertise du système et des données (ou des hypothèses) sur les paramètres de la commande, de la dynamique et de l'environnement. Le moniteur est permissif dans tous les cas expérimentés.

5.5 Conclusion

Nous avons appliqué notre méthode sur un cas d'étude industriel. Nos hypothèses d'architecture du moniteur, de forme des règles de sécurité, comme la problématique de la permissivité ont été confirmés par notre partenaire industriel Kuka. L'analyse de risque Hazop-UML a été conduite sur le système pour en dégager 13 invariants de sécurité. Faute d'observation ou d'intervention, certains invariants n'ont pas pu être modélisés ou implémentés. En particulier, remarquons que le recours aux commandes haptiques est peu compatible avec un moniteur de sécurité.

Sept invariants ont été modélisés, illustrant les dépendances entre variables, la construction d'une marge à partir d'observation booléenne ou encore une observation propre au moniteur comme un timer. Le modèle construit pour l'invariant SI6 illustre l'intérêt de la réponse négative de la synthèse, résolue par l'ajout d'une dépendance dans le modèle. Ces exemples montrent que l'expressivité de la modélisation est suffisante. Cinq stratégies synthétisées ont été implémentées dans un moniteur de sécurité émulé, et testées sur le système réel. Nos tests, loin d'être exhaustifs, montrent que les stratégies sont satisfaisantes mais dépendent fortement des valeurs de marges.

L'application de la méthode montre que si les invariants sont issus de l'analyse de risque, leur modélisation avec des variables observables conduit à en séparer certains. Ainsi, le danger constitué par la collision avec le bras est séparé en deux invariants SI1 et SI8, pour borner à la fois la vitesse du bras seul et la vitesse dans le référentiel cartésien. Les invariants peuvent aussi servir à implémenter des hypothèses utiles pour couvrir d'autres invariants. Par exemple, le seuil d'accélération maximale nécessaire pour que le comportement dans le cas de test 2 soit satisfaisant pourrait être intégré à notre méthode en tant qu'invariant.

Notre méthode a été appliquée avec succès. Au niveau de la modélisation, le principal obstacle rencontré est le manque d'observations ou d'interventions. La complexité de la synthèse a été bien maîtrisée, les temps d'exécutions ne dépassant pas la seconde. L'implémentation a confirmé le caractère crucial du calcul des marges.

Ce qu'il faut retenir :

- L'expressivité de notre modélisation est suffisante pour modéliser les invariants.
- Les expérimentations sont conduites sur un système réel.
- Toutes les stratégies implémentées se sont révélées permissives et sûres vis-à-vis des cas de tests proposés (à l'exception des cas 2 et 4).
- L'efficacité d'un moniteur reste très dépendante de son implémentation et du choix des marges.

Conclusion générale

En conclusion, nous faisons le bilan de nos contributions avant d'ouvrir sur les perspectives de ces travaux.

Bilan

Les systèmes autonomes sont caractérisés par leur polyvalence, leur commande complexe et les interactions avec un environnement peu structuré. Pour en assurer la sécurité, la tolérance aux fautes est donc toute indiquée, et notamment l'approche du moniteur de sécurité séparé. Le comportement du moniteur est régi par des stratégies définies à partir des moyens d'observation et d'intervention propres au moniteur. Notre méthode part des invariants de sécurité résultant de l'analyse de risque et assure que les stratégies sont non seulement sûres mais aussi permissives.

Pour chaque invariant, on construit à partir des observations du moniteur une vision abstraite du comportement du système (et de ses interactions avec l'environnement), des interventions du moniteur et des propriétés de sécurité et de permissivité. L'abstraction s'appuie sur les seuils définis par l'invariant et sur les seuils de marge. La notion de marge, qui s'applique aux variables, permet de définir des états d'alerte. On fixe ainsi la forme des stratégies comme une association entre états d'alerte et interventions. Le recours à une abstraction permet de vérifier les propriétés de sécurité et de permissivité. La construction de l'abstraction, partant du comportement le plus large et le restreignant peu à peu, est conservative vis-vis de la sécurité.

Le modèle ainsi défini contient toutes les informations nécessaires à l'élaboration des stratégies. La synthèse permet de les obtenir automatiquement et de garantir la minimalité des stratégies. Lorsqu'une stratégie a été choisie pour chaque invariant, l'absence de conflit entre les stratégies doit être vérifiée.

Nous avons proposé une formalisation de notre problématique, indépendante de l'encodage dans un outil particulier, et deux approches différentes de synthèse différentes, par la vérification de modèle ou la théorie des jeux. Ainsi, notre méthode peut être appliquée tout en tirant profit des évolutions des outils de méthode formelles. Pour l'heure, nous proposons pour son application des outils s'appuyant sur le vérificateur de modèle NuSMV.

Une étude de cas illustre l'application de cette méthode sur un exemple industriel dans lequel un moniteur est déjà implémenté. Notre approche a permis de combler un manque méthodologique pour déterminer les règles de sécurité. La prise en compte de la permissivité et le lien avec l'analyse de risque ont été des éléments particulièrement appréciés par notre partenaire industriel.

Notre méthode s'insère dans le cycle de développement du système. Elle permet de déterminer, en amont du développement, les états dans lesquels le moniteur sera actif, empêchant ou restreignant le système dans l'accomplissement des tâches.

La commande principale, et en particulier ses fonctions applicatives, peut ainsi être développée pour éviter ces états et les interventions du moniteur perturbant les tâches. Par ailleurs, cela permet d'identifier, parmi tous les moyens possibles d'observation et d'intervention, ceux qui sont suffisants pour assurer la sécurité.

Contributions

Les contributions de ces travaux sont les suivantes :

— Une méthode de l'analyse de risque à l'implémentation, s'insérant entre les itérations sur l'analyse de risque, est proposée.

— Un cadre de modélisation conservatif pour les propriétés de sécurité est formalisé. Il prend explicitement en compte la permissivité et permettant la définition de plusieurs interventions. Le passage par un modèle formel oblige l'utilisateur à expliciter toutes les dépendances et les hypothèses utiles à la sécurité.

— Un algorithme permettant de synthétiser des stratégies et dont les propriétés de complétude et d'exclusivité sont établies. La synthèse complète (en stratégies correctes ou minimales) permet de conclure à la non-existence de solutions et donc à la nécessité de modifier le modèle.

— Des outils (patron de modélisation, outil de complétion et outil de synthèse parallélisé) ont été développés.

— Ces travaux ont été appliqués avec succès à une étude de cas industrielle.

Perspectives

Nous avons élaboré une méthode pour obtenir les stratégies de sécurité, fondée sur les résultats de l'analyse de risque. Tout au long de la modélisation, la validité des résultats des vérifications de sécurité est toujours garantie, si besoin au prix d'une sur-approximation des évolutions possibles du système. Les exigences de permissivité sont au contraire laissées à l'utilisateur qui peut moduler les propriétés proposées par défaut. Pour guider l'utilisateur, les exigences de permissivité pourraient être déterminées à partir de l'analyse de risque Hazop-UML. En effet, cette analyse de risque part des cas d'utilisation UML du système, et contient donc des informations sur les états du système devant être atteints pour accomplir les tâches demandées. Un processus pour les extraire sous une forme directement exploitable rendrait notre méthode plus systématique du point de vue de la permissivité.

Nous avons élaboré une méthode fondée sur l'abstraction des domaines de définition des variables par un faible nombre de parties. Si elle permet une synthèse efficace, cette approche restreint du même coup la précision de modélisation, notamment pour les dépendances, et donc la permissivité. Cette méthode serait avantageusement prolongée par une modélisation plus précise du système comprenant la commande principale et l'environnement. Une fois la sécurité assurée

par la méthode actuelle, cette seconde étape pourrait avoir une modélisation probabiliste du comportement de l'environnement et donc une évaluation très fine de la permissivité, et notamment des espaces dans lesquels la commande est libre d'évoluer.

Nos travaux concernent un moniteur séparé de la commande principale, déclenchant des interventions au niveau physique pour empêcher toute violation des invariants de sécurité. Ils pourraient être étendus à une définition plus large des moniteurs selon deux axes s'inscrivant dans les démarches existantes de hiérarchisation des mécanismes de sécurité et de défense en profondeur.

Le premier serait une surveillance à plusieurs niveaux de l'architecture du système. Dans le second axe, plusieurs niveaux de marges pour chaque invariant seraient définis par l'analyse de risque. Plutôt que des intervalles de valeurs, on entend ici par marge des causes, des signes avant-coureurs de la violation de l'invariant. Ces différents niveaux permettraient de déclencher des interventions de plus en plus dures à mesure que le système se rapprocherait de la frontière de l'invariant.

Les deux axes peuvent être réunis en faisant correspondre chaque niveau de marge à un niveau de l'architecture. Le moniteur que nous avons étudié constituerait alors le niveau ultime de défense, déclenchant sur le dernier niveau de marge des interventions physiques, très perturbatrices pour les tâches du système. Pour tenter de les éviter, les moniteurs insérés dans l'architecture déclencheraient, sur les niveaux de marges en amont, des interventions internes à la commande, donc moins perturbatrices. Les moniteurs seraient donc ordonnés par niveau de marges, chacun déclenchant sur une marge et considérant le franchissement de la marge suivante comme une violation de sécurité. Au sein de l'architecture, les capacités de calcul et les observations plus précises ouvrent de larges possibilités quant aux formes des règles de sécurité. Elles seraient formellement plus complexes qu'un invariant de prédicats à seuils fixes, les moniteurs étant alors implémentés avec des techniques de la vérification d'exécution. Ainsi, une même approche intègrerait l'analyse de risque, différents niveaux de marge, et des interventions au sein de l'architecture, améliorant la sécurité et la permissivité et facilitant la validation des moniteurs multi-niveaux.

Annexe A

Patron de modélisation pour NuSMV

Cette annexe est constitué du patron de modélisation présenté dans la section 3.3.2. Le patron est ici complété pour l'exemple introduit à la section 3.1. Outre les modules déjà présentés, le patron comprend le module `Mem` qui mémorise définitivement la valeur vraie d'un booléen. Le module `Change()` détecte un changement de valeur ; il est appelé par le module `Continuity()`. Ces deux modules permettent de gérer les transitions diagonales par défaut. Cette annexe comprend le modèle complet, après génération et choix d'une stratégie résultant de la synthèse.

Deux modes sont définis dans le patron, encodés par la variable `mode`. Le mode `eval` est utilisé pour déterminer les propriétés de sécurité et permissivité des stratégies sont satisfaites. En particulier, le modèle est en mode `eval` lorsque les propriétés sont vérifiées par l'outil de synthèse. Le mode de suggestion `sugg` permet de construire manuellement et itérativement une stratégie. Il peut être utile lorsque le modèle n'admet pas de stratégie pour trouver le point de blocage. Le mode de suggestion, aussi appelé synthèse interactive, est expliqué en détail dans [Machin 2014]. Un état d'alerte est artificiellement déclaré comme état initial et les propriétés `sugg_myinterv` permettent de déterminer les interventions pertinentes à associer à cet état. Après le choix d'une ou plusieurs interventions pour cet état, la permissivité est évaluée (mode `eval`). Puis, revenant au mode `sugg`, un autre état d'alerte est déclaré comme état initial. Les itérations se poursuivent jusqu'à obtenir la sécurité. Simulé par NuSMV, le mode `sugg` permet également de voir les états suivants possible si une intervention était associée à un état (variable `i_sim` libre dans le module `Interv()`. Cependant, dans ce cas les pré-conditions séquentielles sont ignorées. L'outil de complétion génère des scripts facilitant la synthèse interactive.

Le patron est disponible en ligne à l'adresse `https://www.laas.fr/projects/smof/`.

```
-------------------------------------
-- GENERIC MODULES - DO NOT MODIFY --
-------------------------------------

MODULE Continuity (lower_b, upper_b, init_val, mode)
VAR
        v : lower_b..upper_b;
        c : Change(v);
INIT mode=eval -> v = init_val;
TRANS next(v) = v | next(v) = v + 1 | next(v) = v - 1
-----------------------------------------------
MODULE Change (var)
VAR __c:boolean;
ASSIGN init(__c):=FALSE;
next(__c):= case
        next(var)!=var : TRUE;
        TRUE : FALSE;
esac;
------------------------------
MODULE Mem(bool_var)
VAR __mem : boolean;
ASSIGN init(__mem):=FALSE;
next(__mem):= case
        next(bool_var)=TRUE : TRUE;
        TRUE : __mem;
        esac;
-----------------------------------------------
MODULE Interv(precond, precond_seq, cond, effect, mode)
DEFINE __pre_c := precond;
VAR i_sim : boolean;
INVAR (mode=eval) -> i_sim=FALSE;

VAR activ : boolean;
ASSIGN
init(activ):=FALSE;
next(activ):=case
        mode=sugg : FALSE;
        activ=FALSE & precond_seq & next(cond & precond) : TRUE;
        activ=TRUE & next(cond & precond) : TRUE;
        TRUE : FALSE;
esac;

TRANS activ | (i_sim & precond) -> effect
-----------------------------------------------
MODULE LiveProp (perm_condition, undesired_path, mode, cata)
-- Normal, complete permissiveness requirement
CTLSPEC NAME reach       := (mode=eval ) -> EF (perm_condition & !
    undesired_path)
CTLSPEC NAME uni_reach   := (mode=eval ) -> ( AG (!undesired_path & !
    cata -> EF(perm_condition & !undesired_path)))
CTLSPEC NAME uni_c_reach := (mode=eval ) -> AG (!undesired_path & !
    cata -> EF(perm_condition & !undesired_path & EG (perm_condition)
    ))
```

```
------------------------------
--       MAIN MODULE        --
--        TO MODIFY         --
-- Replace content in < >   --
------------------------------

MODULE main
FROZENVAR mode : {sugg, eval};

--!! Define state variables with their initial values
-- continuous_var:Continuity(lower_bound, upper_bound, init_cond);
-- Continuity var are accessible by name.v
VAR
<pf_vel> : Continuity(0,<2>,<0>,mode);
<arm_pos> : Continuity(0,<1>,<1>, mode);

-- Some constant definitions may be useful
--Ex DEFINE __max_sp :=5;

--!! Model variable dependencies
-- TRANS dependencies: for impossible transitions
-- WARNING a TRANS constraint must have at least one next() operator
--Ex TRANS velocity.v=0 & next(velocity.v)=0 -> next(position.v) =
    position.v (position cannot change if the velocity is in the part
    encoded by 0)
-- INVAR dependencies: for impossible states, combination of state
    variable values

--!! Specify cata with state variables
DEFINE cata:= <pf_vel.v=2 & arm_pos.v=0>;
--Safety property
INVARSPEC NAME safe := mode=eval -> !cata;

--!! Diag or transition with several (and simultaneous) changes of
    independent variables (by default)
-- Those transitions are not taken into account in permissiveness
    analysis and are not permitted when performing a safety interv in
    sugg mode
-- WARNING : if variables (in some value changes) are dependent, the
    transitions must not be "diag" (the expression of __sum_change
    must be tuned to do so)
-- WARNING : if a safety action has effect on 2 var, this particular
    evolution must not be "diag" (the expression of __sum_change must
    be tuned to do so)
-- WARNING : in any case, __sum_change=0 must be equivalent with no
    change in state variables
-- For default use, ie when variables are independent, fill with state
     variables
DEFINE __sum_change :=  toint(<pf_vel>.c.__c)+toint(<arm_pos>.c.__c);
DEFINE diag := __sum_change >= 2;       -- the previous transition is
    diagonal
VAR mdiag : Mem(diag);  --equivalent to the theoretical variable diag
DEFINE IDEM := __sum_change=0;
TRANS cata -> next(cata)=TRUE & next(IDEM)=TRUE
```

```
INVAR cata -> nb_interv=0                -- only needed for sugg and
    simulation

VAR
--!! model interventions
-- interv : Interv(precond stat, precond_seq, condition to trigger in
    eval mode, effect, mode)
<brake> : Interv(<TRUE, pf_vel.v=0, flag_brake, next(pf_vel.v)!=2>,
    mode);
<lock_arm> : Interv(<arm_pos.v=1 , TRUE,  flag_lock_arm, next(arm_pos.
    v)=1>, mode );

---------------------
-- TO BE GENERATED --
---------------------

--DEFINE nb_interv := toint(brake.i_sim)+toint(lock_arm.i_sim);

------ Suggestion mode / Interactive synthesis
--DEFINE flag_brake := FALSE ;
--DEFINE flag_lock_arm := FALSE ;
--INIT (mode=sugg) -> <pf_vel.v=2 & arm_pos.v=1> -- fill with an
    initial state for simulation and the critical state for
    interactive method
---- Properties for suggestion
--CTLSPEC NAME sugg_brake := mode=sugg -> (brake.i_sim -> brake.
    __pre_c & AX !cata)
--CTLSPEC NAME sugg_lock_arm := mode=sugg -> (lock_arm.i_sim ->
    lock_arm.__pre_c & AX !cata)

------ Automatic synthesis
INIT mode = eval;
INVARSPEC NAME valid := (!flag_brake | brake.__pre_c) & (!
    flag_lock_arm | lock_arm.__pre_c)

------ Permissiveness properties
VAR
live_0 : LiveProp(pf_vel.v=1 & arm_pos.v=0, mdiag.__mem, mode, cata);
live_2 : LiveProp(pf_vel.v=0 & arm_pos.v=1, mdiag.__mem, mode, cata);
live_3 : LiveProp(pf_vel.v=1 & arm_pos.v=1, mdiag.__mem, mode, cata);
live_4 : LiveProp(pf_vel.v=2 & arm_pos.v=1, mdiag.__mem, mode, cata);
live_5 : LiveProp(pf_vel.v=0 & arm_pos.v=0, mdiag.__mem, mode, cata);

------ Warning states
DEFINE flag_st1 := arm_pos.v=0 & pf_vel.v=1;
DEFINE flag_st2 := arm_pos.v=1 & pf_vel.v=1;
DEFINE flag_st3 := arm_pos.v=1 & pf_vel.v=2;

--------------------------
-- SYNTHESIZED STRATEGY --
--------------------------

DEFINE flag_lock_arm := flag_st1;
DEFINE flag_brake := flag_st2 | flag_st3;
```

Bibliographie

[Akesson 2006] Knut Akesson, Martin Fabian, Hugo Flordal et Robi Malik. *Supremica : an integrated environment for verification, synthesis and simulation of discrete event systems*. In International Workshop on Discrete Event Systems (WODES), pages 384–385, 2006. (Cité en page 77.)

[Alami 2006] Rachid Alami, Alin Albu-Schäffer, Antonio Bicchi, Rainer Bischoff, Raja Chatila, Alessandro De Luca, Agostino De Santis, Georges Giralt, Jérémie Guiochet, Gerd Hirzinger, Félix Ingrand, Vincenzo Lippiello, Raffaela Mattone, David Powell, Soumen Sen, Bruno Siciliano, Giovanni Tonietti et Luigi Villani. *Safe and dependable physical human-robot interaction in anthropic domains : State of the art and challenges*. In International Conference on Intelligent Robots and Systems (IROS), pages 1–16, 2006. (Cité en page 7.)

[Alexander 2009a] RD Alexander, NJ Herbert et TP Kelly. *The role of the human in an autonomous system*. In International Conference on Systems Safety, pages 1–6, 2009. (Cité en page 7.)

[Alexander 2009b] Robert Alexander, Nicola Herbert et Tim Kelly. *Deriving safety requirements for autonomous systems*. In SEAS DTC Technical Conference, 2009. (Cité en pages 10 et 19.)

[Alur 1998] Rajeev Alur, Thomas A Henzinger, Freddy YC Mang, Shaz Qadeer, Sriram K Rajamani et Serdar Tasiran. *MOCHA : Modularity in model checking*. In International Conference on Computer Aided Verification (CaV), pages 521–525, 1998. (Cité en page 78.)

[Alur 2002] Rajeev Alur, Thomas A Henzinger et Orna Kupferman. *Alternating-time temporal logic*. Journal of the ACM (JACM), vol. 49, no. 5, pages 672–713, 2002. (Cité en page 77.)

[Araiza-Illan 2014] Dejanira Araiza-Illan, Kerstin Eder et Arthur Richards. *Formal verification of control systems' properties with theorem proving*. In International Conference on Control (CONTROL), pages 244–249, 2014. (Cité en page 14.)

[Avižienis 2004] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell et Carl Landwehr. *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pages 11–33, 2004. (Cité en page 3.)

[Behrmann 2007] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen et Didier Lime. *UPPAAL-Tiga : Time for playing games !* In International Conference on Computer Aided Verification (CaV), pages 121–125, 2007. (Cité en page 78.)

[Bengtsson 1996] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson et Wang Yi. *UPPAAL - a tool suite for automatic verification of real-time systems*. In Hybrid Systems III, pages 232–243. LNCS, 1996. (Cité en page 49.)

[Bensalem 2009] Saddek Bensalem, Matthieu Gallien, Félix Ingrand, Imen Kahloul et Thanh-Hung Nguyen. *Toward a more dependable software architecture for autonomous robots*. IEEE Robotics and Automation Magazine, vol. 16, pages 1–11, 2009. (Cité en page 17.)

[Bensalem 2014] Saddek Bensalem, Klaus Havelund et Andrea Orlandini. *Verification and validation meet planning and scheduling*. International Journal on Software Tools for Technology Transfer, vol. 16, pages 1–12, 2014. (Cité en page 14.)

[Black 2009] Jennifer Black et Philip Koopman. *System safety as an emergent property in composite systems*. In International Conference on Dependable Systems and Networks (DSN), pages 369–378, 2009. (Cité en page 19.)

[Böhm 2010] Petr Böhm et Thomas Gruber. *A novel HAZOP study approach in the RAMS analysis of a therapeutic robot for disabled children*. In International Conference on Computer Safety, Reliability and Security (SafeComp), pages 15–27, 2010. (Cité en page 10.)

[Chu 2011] Hoang-Nam Chu. *Test and Evaluation of the Robustness of the Functional Layer of an Autonomous Robot*. Thèse de doctorat, Institut national polytechnique de Toulouse, 2011. (Cité en page 13.)

[Cimatti 2002] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani et Armando Tacchella. *Nusmv 2 : An opensource tool for symbolic model checking*. In International Conference on Computer Aided Verification (CaV), pages 359–364, 2002. (Cité en page 49.)

[Clarke 1986] Edmund M. Clarke, E. Allen Emerson et A. Prasad Sistla. *Automatic verification of finite-state concurrent systems using temporal logic specifications*. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 8, no. 2, pages 244–263, 1986. (Cité en page 46.)

[Crestani 2015] Didier Crestani, Karen Godary-Dejean et Lionel Lapierre. *Enhancing fault tolerance of autonomous mobile robots*. In Journal of Robotics and Autonomous Systems. Elsevier, 2015. (Cité en page 16.)

[Delgado 2004] Nelly Delgado, Ann Q Gates et Steve Roach. *A taxonomy and catalog of runtime software-fault monitoring tools*. Transactions on Software Engineering, vol. 30, no. 12, pages 859–872, 2004. (Cité en page 13.)

[Dogramadzi 2014] Sanja Dogramadzi, Maria Elena Giannaccini, Christopher Harper, Mohammad Sobhani, Roger Woodman et Jiyeon Choung. *Environmental Hazard Analysis-a Variant of Preliminary Hazard Analysis for Autonomous Mobile Robots*. Journal of Intelligent & Robotic Systems, vol. 76, no. 1, pages 73–117, 2014. (Cité en page 11.)

[Durand 2010] Bastien Durand, Karen Godary-Dejean, Lionel Lapierre, Robin Passama et Didier Crestani. *Fault tolerance enhancement using autonomy adaptation for autonomous mobile robots*. In International Conference on Control

and Fault Tolerant Systems (SysTol), pages 24–29, 2010. (Cité en pages 16 et 22.)

[Ertle 2010] Philipp Ertle, Dennis Gamrad, Holger Voos et Dirk Söffker. *Action planning for autonomous systems with respect to safety aspects*. In International Conference on Systems Man and Cybernetics (SMC), pages 2465–2472, 2010. (Cité en pages 15, 21 et 23.)

[Fleury 1997] Sara Fleury, Matthieu Herrb et Raja Chatila. *GenoM : A Tool for the Specification and the Implementation of Operating Modules in a Distributed Robot Architecture*. In International Conference on Intelligent Robots and Systems (IROS), volume 2, pages 842–849, 1997. (Cité en page 7.)

[Fotoohi 2011] Leila Fotoohi et Axel Graser. *Building a safe care-providing robot*. In International Conference on Rehabilitation Robotics (ICORR), pages 1–6, 2011. (Cité en page 21.)

[Goldberg 2005] Allen Goldberg, Klaus Havelund et Conor McGann. *Runtime verification for autonomous spacecraft software*. In Aerospace Conference, 2005, pages 507–516, 2005. (Cité en page 13.)

[Goodloe 2010] Alwyn E Goodloe et Lee Pike. *Monitoring distributed real-time systems : A survey and future directions*, 2010. Rapport technique, NASA/CR-2010-216724. (Cité en pages 13 et 21.)

[Gspandl 2012] Stephan Gspandl, Siegfried Podesser, Michael Reip, Gerald Steinbauer et Máté Wolfram. *A dependable perception-decision-execution cycle for autonomous robots.* In International Conference on Robotics and Automation (ICRA), pages 2992–2998, 2012. (Cité en page 16.)

[Guide ISO/IEC 51 1999] Guide ISO/IEC 51. *Safety aspects — Guidelines for their inclusion in standards*, 1999. Organisation internationale de normalisation (ISO) / Commision Internationale d'Électrotechnique (IEC). (Cité en page 9.)

[Haddadin 2011] Sami Haddadin. *Towards Safe Robots : Approaching Asimov's 1st Law*. Thèse de doctorat, Université technique de Rhénanie-Westphalie, Aix-la-Chapelle, 2011. (Cité en page 8.)

[Haddadin 2012] Sami Haddadin, Simon Haddadin, Augusto Khoury, Tim Rokahr, Sven Parusel, Rainer Burgkart, Antonio Bicchi et Alin Albu-Schäffer. *On making robots understand safety : Embedding injury knowledge into control*. International Journal of Robotics Research, vol. 31, no. 13, pages 1578–1602, 2012. (Cité en page 8.)

[Holzmann 1997] Gerard J Holzmann. *The model checker SPIN*. IEEE Transactions on software engineering, vol. 23, no. 5, pages 279–295, 1997. (Cité en page 49.)

[Horányi 2013] Gergő Horányi, Zoltán Micskei et István Majzik. *Scenario-based Automated Evaluation of Test Traces of Autonomous Systems*. In Workshop on Dependable Embedded and Cyber-physical Systems (DECS) in the International Conference on Computer Safety, Reliability and Security (SafeComp), 2013. (Cité en page 13.)

[Huang 2008] H.-M. Huang. *Autonomy Levels for Unmanned Systems (ALFUS) Framework - Volume I : Terminology*, 2008. Special Publication 1011-I-2.0, National Institute of Standards and Technology (NIST). (Cité en page 5.)

[IEC61508-7 2010] IEC61508-7. *Sécurité fonctionnelle des systèmes électriques/électroniques/électroniques programmables relatifs à la sécurité - Partie 7 : Présentation de techniques et mesures*, 2010. Commission électotechnique internationale (IEC). (Cité en page 25.)

[ISO10218-1 2006] ISO10218-1. *Robots pour environnements industriels - Exigences de sécurité - Partie 1 : Robot*, 2006. Organisation internationale de normalisation (ISO). (Cité en page 8.)

[ISO13482 2014] ISO13482. *Robots and robotic devices - Safety requirements for personal care robots*, 2014. Organisation internationale de normalisation (ISO). (Cité en page 8.)

[Kane 2014] Aaron Kane, Thomas Fuhrman et Philip Koopman. *Monitor Based Oracles for Cyber-Physical System Testing : Practical Experience Report*. In International Conference on Dependable Systems and Networks (DSN), pages 148–155, 2014. (Cité en page 13.)

[Kruse 2013] Thibault Kruse, Amit Kumar Pandey, Rachid Alami et Alexandra Kirsch. *Human-aware robot navigation : A survey*. Journal of Robotics and Autonomous Systems, vol. 61, no. 12, pages 1726–1743, 2013. (Cité en page 8.)

[Laprie 1996] Jean-Claude Laprie, Jean Arlat, Jean-Paul Blanquart, Alain Costes, Yves Crouzet, Yves Deswarte, Jean-Charles Fabre, Hubert Guillermain, Mohamed Kaâniche, Karama Kanoun, Corinne Mazet, David Powell, Christophe Rabéjac et Pascale Thévenod. *Guide de la sûreté de fonctionnement*. Cépaduès-éditions, 1996. (Cité en page 3.)

[Leucker 2009] Martin Leucker et Christian Schallhart. *A brief account of runtime verification*. Journal of Logic and Algebraic Programming, vol. 78, no. 5, pages 293–303, 2009. (Cité en page 13.)

[Lussier 2007a] Benjamin Lussier. *Tolérance aux fautes dans les systèmes autonomes*. Thèse de doctorat, Institut national polytechnique de Toulouse, 2007. (Cité en pages 6 et 16.)

[Lussier 2007b] Benjamin Lussier, Matthieu Gallien, Jérémie Guiochet, Félix Ingrand, Marc-Olivier Killijian et David Powell. *Fault tolerant planning for critical robots*. In International Conference on Dependable Systems and Networks (DSN), pages 144–153, 2007. (Cité en page 16.)

[Machin 2013] Mathilde Machin, Jérémie Guiochet, David Powell et Hélène Waeselynck. *Introduction à la synthèse de superviseur*. LAAS-CNRS 13067, 2013. (Cité en page 77.)

[Machin 2014] Mathilde Machin, Fanny Dufossé, Jean-Paul Blanquart, Jérémie Guiochet, David Powell et Hélène Waeselynck. *Specifying Safety Monitors*

for Autonomous Systems Using Model-Checking. In International Conference on Computer Safety, Reliability and Security (SafeComp), pages 262–277, 2014. (Cité en pages 54 et 103.)

[Machin 2015a] Mathilde Machin, Fanny Dufossé, Jérémie Guiochet, David Powell, Matthieu Roy et Hélène Waeslynck. *Model-Checking and Game theory for Synthesis of Safety Rules*. In International Symposium on High Assurance Systems Engineering (HASE), pages 36–43, 2015. (Cité en page 80.)

[Machin 2015b] Mathilde Machin, Jérémie Guiochet, Tim Guhl, Steffen Walther et Vito Magnanimo. *SAPHARI D1.3.1. Report on safety monitoring framework and safe control strategies*. LAAS-CNRS 15009, 2015. (Cité en page 85.)

[Mallet 2010] Anthony Mallet, Cédric Pasteur, Matthieu Herrb, Séverin Lemaignan et Félix Ingrand. *GenoM3 : Building middleware-independent robotic components*. In International Conference on Robotics and Automation (ICRA), pages 4627–4632, 2010. (Cité en page 7.)

[Malm 2010] Timo Malm, Juhani Viitaniemi, Jyrki Latokartano, Salla Lind, Outi Venho-Ahonen et Jari Schabel. *Safety of interactive robotics : Learning from accidents*. International Journal of Social Robotics, vol. 2, no. 3, pages 221–227, 2010. (Cité en page 21.)

[Martin-Guillerez 2010] Damien Martin-Guillerez, Jérémie Guiochet, David Powell et Christophe Zanon. *A UML-based method for risk analysis of human-robot interactions*. In International Workshop on Software Engineering for Resilient Systems (SERENE), pages 32–41, 2010. (Cité en pages 11, 12, 119 et 121.)

[Mekki-Mokhtar 2012a] Amina Mekki-Mokhtar. *Processus d'identification de propriétés de sécurité-innocuité vérifiables en ligne pour des systèmes autonomes critiques*. Thèse de doctorat, Université Paul Sabatier, Toulouse, 2012. (Cité en pages 18, 24, 29, 76 et 119.)

[Mekki-Mokhtar 2012b] Amina Mekki-Mokhtar, Jean-Paul Blanquart, Jérémie Guiochet, David Powell et Matthieu Roy. *Safety Trigger Conditions for Critical Autonomous Systems*. In Pacific Rim international symposium on Dependable Computing (PRDC), pages 61–69, 2012. (Cité en page 18.)

[Micskei 2012] Zoltán Micskei, Zoltán Szatmári, János Oláh et István Majzik. *A concept for testing robustness and safety of the context-aware behaviour of autonomous systems*. In Conference on Agent and Multi-Agent Systems. Technologies and Applications (AMSTA), pages 504–513. Springer, 2012. (Cité en page 13.)

[O'Brien 2014] Matthew O'Brien, Ronald C Arkin, Dagan Harrington, Damian Lyons et Shu Jiang. *Automatic Verification of Autonomous Robot Missions*. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), pages 462–473. Springer, 2014. (Cité en page 14.)

[Papadopoulos 2001] Yiannis Papadopoulos et JA McDermid. *Automated safety monitoring : A review and classification of methods*. International Journal of Condition Monitoring and Diagnostic Engineering Management (COMADEM), vol. 4, no. 4, pages 14–32, 2001. (Cité en page 21.)

[Pathak 2013] Shashank Pathak, Luca Pulina, Giorgio Metta et Armando Tacchella. *Ensuring safety of policies learned by reinforcement : Reaching objects in the presence of obstacles with the iCub*. In International Conference on Intelligent Robots and Systems (IROS), pages 170–175, 2013. (Cité en page 14.)

[Pike 2012] Lee Pike, Sebastian Niller et Nis Wegmann. *Runtime verification for ultra-critical systems*. In International Conference on Runtime Verification (RV), pages 310–324, 2012. (Cité en pages 20 et 22.)

[Pnueli 1977] Amir Pnueli. *The temporal logic of programs*. In 18th Annual Symposium on Foundations of Computer Science, pages 46–57. IEEE, 1977. (Cité en page 46.)

[Powell 2012] David Powell, Jean Arlat, Hoang Nam Chu, Félix Ingrand et Marc-Olivier Killijian. *Testing the input timing robustness of real-time control software for autonomous systems*. In European Dependable Computing Conference (EDCC), pages 73–83, 2012. (Cité en page 13.)

[Py 2004a] Frédéric Py. *Contrôle d'Eeécution dans une architecture hiérarchisée pour systèmes autonomes*. Thèse de doctorat, Université Paul Sabatier, Toulouse, 2004. (Cité en page 16.)

[Py 2004b] Frédéric Py et Félix Ingrand. *Dependable execution control for autonomous robots*. In International Conference on Intelligent Robots and Systems (IROS), pages 1136–1141, 2004. (Cité en pages 16 et 26.)

[Quigley 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler et Andrew Y Ng. *ROS : an open-source Robot Operating System*. International Conference on Robotics and Automation (ICRA), Workshop on open source software, vol. 3, no. 3.2, pages 5–11, 2009. (Cité en page 7.)

[Ramadge 1987] Peter J Ramadge et W Murray Wonham. *Supervisory control of a class of discrete event processes*. Society for Industrial and Applied Mathematics (SIAM) journal on control and optimization, vol. 25, no. 1, pages 206–230, 1987. (Cité en page 21.)

[Schaeffer 2013] Kristin Schaeffer. *The perception and measurement of human-robot trust*. Thèse de doctorat, Université de Floride centrale, Orlando, 2013. (Cité en page 8.)

[Scherer 2005] Sebastian Scherer, Flavio Lerda et Edmund M Clarke. *Model checking of robotic control systems*. In International Symposium on Artificial Intelligence, Robotics and Automation in Space (SAIRAS), 2005. (Cité en page 14.)

[Schnoebelen 1999] Philippe Schnoebelen, Béatrice Bérard, Michel Bidoit, François Laroussinie et Antoine Petit. *Vérification de logiciels : techniques et outils du model-checking*. Vuibert, Avril 1999. (Cité en page 46.)

[Sha 2001] Lui Sha. *Using simplicity to control complexity*. IEEE Journal Software, vol. 18, no. 4, pages 20–28, 2001. (Cité en page 21.)

[Simmons 2000] Reid Simmons, Charles Pecheur et Grama Srinivasan. *Towards automatic verification of autonomous systems*. In International Conference on Intelligent Robots and Systems (IROS), volume 2, pages 1410–1415, 2000. (Cité en page 14.)

[Sisbot 2012] Emrah Akin Sisbot et Rachid Alami. *A human-aware manipulation planner*. Transactions on Robotics,, vol. 28, no. 5, pages 1045–1057, 2012. (Cité en page 8.)

[Steinbauer 2013] Gerald Steinbauer. *A Survey about Faults of Robots used in RoboCup*. In RoboCup 2012 : Robot Soccer World Cup XVI, pages 344–355. Springer, 2013. (Cité en page 9.)

[Täubig 2012] Holger Täubig, Udo Frese, Christoph Hertzberg, Christoph Lüth, Stefan Mohr, Elena Vorobev et Dennis Walter. *Guaranteeing functional safety : design for provability and computer-aided verification*. Journal Autonomous Robots, vol. 32, no. 3, pages 303–331, 2012. (Cité en page 14.)

[Tomatis 2003] Nicola Tomatis, Gregoire Terrien, Ralph Piguet, Daniel Burnier, Samir Bouabdallah, Kai Oliver Arras et Roland Siegwart. *Designing a secure and robust mobile interacting robot for the long term*. In International Conference on Robotics and Automation (ICRA), pages 4246–4251, 2003. (Cité en pages 17 et 20.)

[Tonietti 2005] Giovanni Tonietti, Riccardo Schiavi et Antonio Bicchi. *Design and control of a variable stiffness actuator for safe and fast physical human/robot interaction*. In International Conference on Robotics and Automation (ICRA), pages 526–531, 2005. (Cité en page 7.)

[Wonham 2005] W. Murray Wonham. *Supervisory control of discrete event systems*, 2005. (Cité en pages 21 et 77.)

[Woodman 2012] Roger Woodman, Alan F.T. Winfield, Chris Harper et Mike Fraser. *Building safer robots : Safety driven control*. Internatioanl Journal of Robotics Research, vol. 31, no. 13, pages 1603–1626, 2012. (Cité en pages 10, 18 et 26.)

[Woodman 2013] Roger Woodman. *Novel approaches for the safety of human-robot interaction*. Thèse de doctorat, Université de l'Ouest de l'Angleterre, Bristol, 2013. (Cité en pages 18 et 23.)

[Zou 2014] Xueyi Zou, Rob Alexander et John McDermid. *Safety Validation of Sense and Avoid Algorithms Using Simulation and Evolutionary Search*. In International Conference on Computer Safety, Reliability, and Security (SafeComp), pages 33–48, 2014. (Cité en page 13.)

Table des matières

Table des figures

Liste des tableaux

Résumé : Les systèmes autonomes, notamment ceux opérant à proximité d'êtres humains, soulèvent des problèmes de sécurité-innocuité puisqu'ils peuvent causer des blessures. La complexité de la commande de ces systèmes, ainsi que leurs interactions avec un environnement peu structuré, rendent difficile l'élimination complète des fautes. Nous adoptons donc une démarche de tolérance aux fautes en considérant un moniteur de sécurité séparé de la commande principale et qui dispose de ses propres moyens d'observation et d'intervention. Le comportement de ce moniteur est régi par des règles qui doivent assurer la sécurité du système tout en lui permettant de remplir ses fonctions. Nous proposons une méthode systématique pour obtenir ces règles de sécurité. Les dangers, déterminés par une analyse de risque, sont modélisés formellement puis un algorithme synthétise des règles sûres et permissives, s'il en existe. Nous avons outillé cette méthode pour les étapes de modélisation et de synthèse en nous appuyant sur l'outil de vérification de modèle NuSMV. L'étude d'un cas industriel illustre l'application de la méthode et des outils sur un robot manufacturier dans un environnement humain.

Mots clés : Moniteur de sécurité, Règles de sécurité, Systèmes autonomes, Tolérance aux fautes, Vérification, Synthèse.

Abstract: Autonomous systems operating in the vicinity of humans are critical in that they potentially harm humans. In these systems, fault removal is not sufficient given the command complexity and their interactions with an unstructured environment. By a fault tolerance approach, we consider a safety monitor separated from the main command and able to observe and intervene on the system. The monitor behavior is specified by safety rules that must both ensure safety and permit the system to carry out its tasks in absence of hazard. We propose a systematic method to obtain these safety rules. The hazards, determined by a risk analysis, are formally modeled, then an algorithm synthesizes safe and permissive rules, if any exists. The method is tooled both for modeling and synthesis by use of the model-checker NuSMV. Method and tools are applied to the industrial use case of a robotic co-worker.

Key-words: Safety monitor, Safety rules, Autonomous systems, Fault tolerance, Verification, Synthesis.